Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (Extended Version 1, April 24th, 2012)

Benedikt Schmidt, Simon Meier, Cas Cremers, David Basin
Institute of Information Security, ETH Zurich, Switzerland

***Abstract*—We present a general approach for the symbolic analysis of security protocols that use Diffie-Hellman exponentiation to achieve advanced security properties. We model protocols as multiset rewriting systems and security properties as first-order formulas. We analyze them using a novel constraint-solving algorithm that supports both falsification and verification, even in the presence of an unbounded number of protocol sessions. The algorithm exploits the finite variant property and builds on ideas from strand spaces and proof normal forms. We demonstrate the scope and the effectiveness of our algorithm on non-trivial case studies. For example, the algorithm successfully verifies the NAXOS protocol with respect to a symbolic version of the eCK security model.**

This is the extended version of [1]. It contains additional explanations and the proofs justifying the results in [1].

I. Introduction

Authenticated Key Exchange (AKE) protocols are widely used components in modern network infrastructures. They assume a Public-Key Infrastructure and use the public keys to establish shared session keys over an untrusted channel. Recent AKE protocols use Diffie-Hellman (DH) exponentiation to achieve advanced security properties, namely secrecy and authentication properties in the presence of adversaries who are significantly more powerful than the classical Dolev-Yao adversary. For example, in the eCK model [2], the adversary may corrupt random number generators and dynamically compromise long-term keys and session keys.

As witnessed by the numerous attacks on published protocols, e.g. [3]–[6], designing AKE protocols is error-prone. It is therefore desirable to formally verify them before deployment, ideally automatically and with respect to an unbounded number of sessions. In this paper, we use a symbolic model of DH exponentiation to enable automatic verification. Our model supports DH exponentiation and an abelian group of exponents. This allows the adversary to cancel out DH exponents using exponentiation with their inverse. Similar to previous work on automatic symbolic analysis [7]–[9], we do not model multiplication in the DH group and addition of exponents.

There are no existing approaches capable of automatically verifying recent AKE protocols in models combining advanced security properties, unbounded sessions, and DH exponentiation. Existing approaches either bound the number of sessions [8], [9], fail to model the required adversary capabilities [7], [10]–[12], do not consider inverses in the group of DH exponents [13]–[15], or faithfully model the adversary, but do not support DH exponentiation [16], [17]. In this paper, we give a general approach to security protocol verification, which is capable of automatically verifying AKE protocols in models as described above.

Contributions: First, we give an expressive and general security protocol model, which uses multiset rewriting to specify protocols and adversary capabilities, a guarded fragment [18] of first-order logic to specify security properties, and equational theories to model the algebraic properties of cryptographic operators.

Second, we give a novel constraint-solving algorithm for the falsification and verification of security protocols specified in our model for an unbounded number of sessions. We give a full proof of its correctness along with proofs of all theorems and assertions in this paper in Appendix C.

Third, we implemented our algorithm in a tool, the Tamarin prover [19], and validated its effectiveness on a number of non-trivial case studies. Despite the undecidability of the verification problem, our algorithm performs well: it terminates in the vast majority of cases, and the times for falsification and verification are in the range of a few seconds. This makes Tamarin well-suited for the automated analysis of security protocols that use DH exponentiation to achieve advanced security properties.

Organization: We introduce notation in Section II and provide background on the security properties of AKE protocols in Section III. In Section IV, we define our protocol model. We present the theory underlying our constraint-solving algorithm in Section V and the algorithm in Section VI. We perform case studies in Section VII, compare with related work in Section VIII, and conclude in Section IX.

II. Notational Preliminaries

S^* denotes the set of sequences over S. For a sequence s, we write s_i for the i-th element, $|s|$ for the length of s, and $idx(s) = \{1, \dots, |s|\}$ for the set of indices of s. We write $\vec{s}$ to emphasize that s is a sequence. We use $[]$ to denote the empty sequence, $[s_1, \dots, s_k]$ to denote the sequence s where $|s| = k$, and $s \cdot s'$ to denote the concatenation of the sequences s and s'. $S^\sharp$ denotes the set of finite multisets with elements from S. We also use the superscript $\sharp$ to denote the usual operations on multisets such as $\cup^\sharp$. For a sequence s,

$$\begin{array}{ccc} \mathcal{I} & & \mathcal{R} \\ \text{create fresh } esk_{\mathcal{I}} & \xrightarrow{g^{h_1(esk_{\mathcal{I}}, lk_{\mathcal{I}})}} & \text{receive } X \\ \text{receive } Y & \xleftarrow{g^{h_1(esk_{\mathcal{R}}, lk_{\mathcal{R}})}} & \text{create fresh } esk_{\mathcal{R}} \end{array}$$

$$\begin{array}{rllllll} k_{\mathcal{I}} = & h_2(Y^{lk_{\mathcal{I}}}, & x, & Y^{h_1(esk_{\mathcal{I}}, lk_{\mathcal{I}})}, & \mathcal{I}, & \mathcal{R}) \\ k_{\mathcal{R}} = & h_2(y, & X^{lk_{\mathcal{R}}}, & X^{h_1(esk_{\mathcal{R}}, lk_{\mathcal{R}})}, & \mathcal{I}, & \mathcal{R}) \end{array}$$
$$\text{for } y = (pk_{\mathcal{I}})^{h_1(esk_{\mathcal{R}}, lk_{\mathcal{R}})} \text{ and } x = (pk_{\mathcal{R}})^{h_1(esk_{\mathcal{I}}, lk_{\mathcal{I}})}$$

Figure 1. The NAXOS protocol.

mset(s) denotes the corresponding multiset and *set*(s) the corresponding set. We also use *set*(m) for multisets m.

We write *vars*(t) for the set of all variables in t, and *fvars*(F) for the set of all variables that have free occurrences in a formula F. For a function f, we write $f[a \mapsto b]$ to denote the function that maps a to b and c to $f(c)$, for all $c \neq a$.

III. Authenticated Key Exchange Protocols

We use the NAXOS protocol [2] as an example to illustrate the constructions and goals underlying recent AKE protocols. Figure 1 depicts the protocol. Each party x has a long-term private key lk_x and a corresponding public key $pk_x = g^{lk_x}$, where g is a generator of the DH group. To start a session, the initiator $\mathcal{I}$ first creates a fresh nonce $esk_{\mathcal{I}}$, also known as $\mathcal{I}$'s ephemeral (private) key. He then concatenates $esk_{\mathcal{I}}$ with $\mathcal{I}$'s long-term private key $lk_{\mathcal{I}}$, hashes the result using h_1, and sends $g^{h_1(esk_{\mathcal{I}}, lk_{\mathcal{I}})}$ to the responder. The responder $\mathcal{R}$ stores the received value in a variable X, computes a similar value based on his own nonce $esk_{\mathcal{R}}$ and long-term private key $lk_{\mathcal{R}}$, and sends the result to the initiator, who stores the received value in the variable Y. Finally, both parties compute a session key ($k_{\mathcal{I}}$ and $k_{\mathcal{R}}$, respectively) whose computation includes their own long-term private keys, such that only the intended partner can compute the same key.

Note that the messages exchanged are not authenticated, as the recipients cannot verify that the expected long-term key was used in the construction of the message. The authentication is implicit and only guaranteed through ownership of the correct key. Explicit authentication (e.g., the intended partner was recently alive or agrees on some values) is commonly achieved in AKE protocols by adding a key-confirmation step, where the parties exchange a MAC of the exchanged messages that is keyed with (a variant of) the computed session key.

The key motivation behind recent AKE protocols is that they should achieve their security goals even in the presence of very strong adversaries. For example, the NAXOS protocol is designed to be secure in the eCK security model [2]. In this model, as in the standard Dolev-Yao model, the adversary has complete control over the network and can learn the long-term private keys of all dishonest agents. However, unlike in the Dolev-Yao model, he can additionally, under some restrictions, learn the long-term private key of any agent. This models (weak) *Perfect Forward Secrecy* (wPFS/PFS): even if the adversary learns the long-term private keys of all the agents, the keys of previous sessions should remain secret [20]. Additionally, this models resilience against *Key Compromise Impersonation* (KCI): even if the adversary learns the long-term private key of an agent, he should be unable to impersonate as anybody to this agent [6]. Moreover, the adversary can learn the session keys of certain sessions. This models both Key Independence (KI), where compromising one session key should not compromise other keys, and resilience against *unknown-key share attacks* (UKS), where the adversary should not be able to trick other sessions into computing the same key. Finally, the adversary can learn any agent's ephemeral keys. This models resilience against corrupted random-number generators. All these attack types are modeled in the eCK security model.

We call security properties that consider such strong adversaries *advanced security properties*. We give an example of such a property by formalizing the security of the NAXOS protocol in the eCK model in Section IV-C.

IV. Security Protocol Model

We model the execution of a security protocol in the context of an adversary as a labeled transition system, whose state consists of the adversary's knowledge, the messages on the network, information about freshly generated values, and the protocol's state. The adversary and the protocol interact by updating network messages and freshness information. Adversary capabilities and protocols are specified jointly as a set of (labeled) multiset rewriting rules. Security properties are modeled as trace properties of the transition system.

In the following, we first describe how protocols are specified and executed. Then, we define our property specification language and illustrate our protocol model with an example.

A. Protocol Specification and Execution

To model cryptographic messages, we use an order-sorted term algebra with the sort *msg* and two incomparable subsorts *fresh* and *pub* for fresh and public names. We assume there are two countably infinite sets *FN* and *PN* of *fresh* and *public names* and a countably infinite set $\mathcal{V}_s$ of variables for each sort s. We denote the union of these $\mathcal{V}_s$ by $\mathcal{V}$. We write $x{:}s$ to denote that $x \in \mathcal{V}_s$. Our approach supports a user-defined signature for modeling cryptographic operators other than DH exponentiation. To simplify its presentation, we use however a fixed signature with the function symbols

$$\Sigma_{DH} = \{\mathrm{enc}(\_,\_), \mathrm{dec}(\_,\_), \mathrm{h}(\_), \langle\_,\_\rangle, \mathrm{fst}(\_), \mathrm{snd}(\_), \\ \_\hat{\ }\_, \_^{-1}, \_*\_, 1\} \ ,$$

which are all of sort $msg \times \ldots \times msg \to msg$. The symbols in the first line model symmetric encryption, hashing, and pairing. Those in the second line model DH exponentiation

(1)	$\mathrm{dec}(\mathrm{enc}(m,k),k) \simeq m$	(6)	$x * 1 \simeq x$
(2)	$\mathrm{fst}(\langle x,y\rangle) \simeq x$	(7)	$x * x^{-1} \simeq 1$
(3)	$\mathrm{snd}(\langle x,y\rangle) \simeq y$	(8)	$(x^{-1})^{-1} \simeq x$
(4)	$x * (y * z) \simeq (x * y) * z$	(9)	$(x \hat{} y) \hat{} z \simeq x \hat{} (y * z)$
(5)	$x * y \simeq y * x$	(10)	$x \hat{} 1 \simeq x$

Figure 2. Equations that constitute E_{DH}.

and inversion, multiplication, and the unit in the group of exponents.

We abbreviate the set of well-sorted terms built over Σ_{DH}, *PN*, *FN*, and $\mathcal{V}$ as $\mathcal{T}$. Cryptographic *messages* are modeled by the ground terms in $\mathcal{T}$, which we abbreviate as $\mathcal{M}$. In the remainder of the paper, we use g to denote a public name that is used as a fixed generator of the DH group and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{k}$ to denote fresh names.

The equational theory E_{DH} generated by the equations in Figure 2 formalizes the semantics of the function symbols in Σ_{DH}. It consists of equations for decryption and projection (1–3), exponentiation (9–10), and the theory of abelian groups for the exponents (4–8). Equation (9) states that repeated exponentiation in a DH group corresponds to multiplication of the exponents.

As an example, consider the term $((\mathsf{g} \hat{} \mathsf{a}) \hat{} \mathsf{b}) \hat{} \mathsf{a}^{-1}$, which results from exponentiating g with a, followed by b, followed by a inverse. This is equal to $\mathsf{g} \hat{} ((\mathsf{a} * \mathsf{b}) * \mathsf{a}^{-1})$ because of (9) and can be further simplified to $\mathsf{g} \hat{} \mathsf{b}$ using (4–7).

Note that our approach supports the combination of Equations (2–10) modeling DH exponentiation and pairing with an arbitrary subterm-convergent rewriting theory for the user-defined cryptographic operators (see Appendix C). A rewriting theory R is subterm-convergent if it is convergent and for each rule $l \to r \in R$, r is either a proper subterm of l or is ground and in normal form with respect to R. One can therefore extend Σ_{DH} and E_{DH} with asymmetric encryption, signatures, and similar operators.

Note that our equational theory does not support protocols that perform multiplication in the DH group G. To define such protocols, an additional function symbol $\times$ denoting multiplication in G is required. The function symbol $*$ denotes multiplication in the group of exponents, which is a different operation. For example, the equality $(\mathsf{g}\hat{}\mathsf{a} \times \mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c} = (\mathsf{g}\hat{}\mathsf{a})\hat{}\mathsf{c} \times (\mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c}$ holds in all DH groups, but does usually not hold if we replace $\times$ by $*$. Moreover, addition of exponents must be modeled for such protocols to avoid missing attacks. Consider the example protocol that randomly choses two exponents a and b, sends these exponents, receives some exponent x, and checks if $\mathsf{g} \hat{} \mathsf{a} \times \mathsf{g} \hat{} \mathsf{b} = \mathsf{g} \hat{} \mathsf{x}$. This check succeeds if and only if $\mathsf{x} = \mathsf{a} + \mathsf{b}$.

1) Transition System State: We model the states of our transition system as finite multisets of facts. We use a fixed set of fact symbols to encode the adversary's knowledge, freshness information, and the messages on the network. The remaining fact symbols are used to represent the protocol state. Formally, we assume an unsorted signature Σ_{Fact} partitioned into *linear* and *persistent* fact symbols. We define the set of *facts* as the set $\mathcal{F}$ consisting of all facts $F(t_1, .., t_k)$ such that $t_i \in \mathcal{T}$ and $F \in \Sigma^k_{Fact}$. We denote the set of *ground facts* by $\mathcal{G}$. We say that a fact $F(t_1, .., t_k)$ is *linear* if F is linear and *persistent* if F is persistent.

Linear facts model resources that can only be consumed once, whereas persistent facts model inexhaustible resources that can be consumed arbitrarily often. In the rest of the paper, we assume that Σ_{Fact} consists of an arbitrary number of protocol-specific fact symbols to describe the protocol state and the following special fact symbols. A persistent fact $\mathsf{K}(m)$ denotes that m is known to the adversary. A linear fact $\mathsf{Out}(m)$ denotes that the protocol has sent the message m, which can be received by the adversary. A linear fact $\mathsf{In}(m)$ denotes that the adversary has sent the message m, which can be received by the protocol. A linear fact $\mathsf{Fr}(n)$ denotes that the fresh name n was freshly generated.

2) Adversary, Protocol, and Freshness Rules: To specify the possible transitions by the adversary and the honest participants, we use *labeled multiset rewriting*. A *labeled multiset rewriting rule* is a triple (l, a, r) with $l, a, r \in \mathcal{F}^*$, denoted $l\!-\![\, a \,]\!\!\rightarrow r$. We often suppress the brackets around the sequences l, a, and r when writing rules. For $ri = l\!-\![\, a \,]\!\!\rightarrow r$, we define the *premises* as $prems(ri) = l$, the *actions* as $acts(ri) = a$, and the *conclusions* as $concs(ri) = r$. We use $ginsts(R)$ to denote the *set of ground instances* of a set of labeled multiset rewriting rules R.

There are three types of rules. A rule for fresh name generation, the message deduction rules, and the rules specifying the protocol and the adversary's capabilities. All fresh names are created with the rule $\textsc{Fresh} = ([\,]\!-\![\,]\!\!\rightarrow \mathsf{Fr}(x{:}fresh))$. This is the only rule that produces Fr facts and we consider only runs with unique instances of this rule, i.e., the same fresh name is never generated twice.

We use the following set of *message deduction rules*.

$$\begin{aligned} MD = \{\, & \mathsf{Out}(x)\!-\![\,]\!\!\rightarrow \mathsf{K}(x), \quad \mathsf{K}(x)\!-\![\, \mathsf{K}(x) \,]\!\!\rightarrow \mathsf{In}(x) \,\} \\ \cup \{\, & -\![\,]\!\!\rightarrow \mathsf{K}(x{:}pub), \quad \mathsf{Fr}(x{:}fresh)\!-\![\,]\!\!\rightarrow \mathsf{K}(x{:}fresh) \,\} \\ \cup \{\, & \mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)\!-\![\,]\!\!\rightarrow \mathsf{K}(f(x_1, \ldots, x_k)) \mid f \in \Sigma^k_{DH} \,\} \end{aligned}$$

The rules in the first line allow the adversary to receive messages from the protocol and send messages to the protocol. The $\mathsf{K}(x)$ action in the second rule makes the messages sent by the adversary observable in a protocol's trace. We exploit this to specify secrecy properties. The rules in the second line allow the adversary to learn public names and freshly generated names. The remaining rules allow the adversary to apply functions from Σ_{DH} to known messages.

A *protocol rule* is a multiset rewriting rule $l\!-\![\, a \,]\!\!\rightarrow r$ such that **(P1)** it does not contain fresh names, **(P2)** K and Out facts do not occur in l, **(P3)** K, In, and Fr facts do not occur in r, and **(P4)** $vars(r) \subseteq vars(l) \cup \mathcal{V}_{pub}$. A *protocol* is a finite set of protocol rules. Note that our formal notion of a protocol encompasses both the rules executed by the honest participants and the adversary's capabilities, like revealing long-term keys. Condition **P1** and the restriction on the usage of Fr facts from **P3**, which also hold for the message deduction rules, ensure that all fresh names originate from instances of the FRESH rule.

3) Transition Relation: The labeled transition relation $\rightarrow_P \subseteq \mathcal{G}^\sharp \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^\sharp$ for a protocol P is defined by the transition rule

$$\frac{l \text{-}[\ a\ ]\!\!\rightarrow r \in_{E_{DH}} \mathit{ginsts}(P \cup \mathit{MD} \cup \{\textsc{Fresh}\}) \qquad \mathit{lfacts}(l) \subseteq^\sharp S \quad \mathit{pfacts}(l) \subseteq \mathit{set}(S)}{S \xrightarrow{\mathit{set}(a)}_P ((S \setminus^\sharp \mathit{lfacts}(l)) \cup^\sharp \mathit{mset}(r))} \ ,$$

where $\mathit{lfacts}(l)$ is the multiset of all linear facts in l and $\mathit{pfacts}(l)$ is the set of all persistent facts in l. This transition rule models rewriting the state with a ground instance of a protocol rule, a message deduction rule, or the FRESH rule. Since we perform multiset rewriting modulo E_{DH}, we use $\in_{E_{DH}}$ for the rule instance. As linear facts are consumed upon rewriting, we use multiset inclusion to check that all facts in $\mathit{lfacts}(l)$ occur sufficiently many times in S. For persistent facts, we only check that every fact in $\mathit{pfacts}(l)$ occurs in S. To obtain the successor state, we remove the consumed linear facts and add the generated facts. The action associated to the transition is the set of actions of the rule instance.

A *trace* is a sequence of sets of ground facts denoting the sequence of actions that happened during a protocol's execution. We model the executions of a security protocol P by its set of traces defined as

$$\begin{aligned} \mathit{traces}(P) = \{[A_1, \dots, A_n] \\ \mid \exists S_1, \dots, S_n \in \mathcal{G}^\sharp.\ \varnothing^\sharp \xrightarrow{A_1}_P \dots \xrightarrow{A_n}_P S_n \\ \wedge \forall i \neq j.\ \forall x.\ (S_{i+1} \setminus^\sharp S_i) = \{\mathsf{Fr}(x)\}^\sharp \Rightarrow \\ (S_{j+1} \setminus^\sharp S_j) \neq \{\mathsf{Fr}(x)\}^\sharp\ \}\ . \end{aligned}$$

The second conjunct ensures that each instance of the FRESH rule is used at most once in a trace. Each consumer of a $\mathsf{Fr}(n)$ fact therefore obtains a different fresh name. Transitions labeled with $\varnothing$ are silent. We therefore define the *observable trace* $\overline{tr}$ of a trace tr as the subsequence of all non-silent actions in tr.

B. Security Properties

We use two-sorted first-order-logic to specify security properties. This logic supports quantification over both messages and timepoints. We thus introduce the sort *temp* for timepoints and write $\mathcal{V}_{temp}$ for the set of *temporal variables*.

A *trace atom* is either *false* $\bot$, a *term equality* $t_1 \approx t_2$, a *timepoint ordering* $i \lessdot j$, a *timepoint equality* $i \doteq j$, or an *action* $f@i$ for a fact f and a timepoint i. A *trace formula* is a first-order formula over trace atoms.

To define the semantics of trace formulas, we associate a domain $\mathbf{D}_s$ with each sort s. The domain for temporal variables is $\mathbf{D}_{temp} = \mathbb{Q}$ and the domains for messages are $\mathbf{D}_{msg} = \mathcal{M}$, $\mathbf{D}_{fresh} = \mathit{FN}$, and $\mathbf{D}_{pub} = \mathit{PN}$. We say a function θ from $\mathcal{V}$ to $\mathbb{Q} \cup \mathcal{M}$ is a *valuation* if it respects sorts, i.e., $\theta(\mathcal{V}_s) \subseteq \mathbf{D}_s$ for all sorts s. For a term t, we write $t\theta$ for the application of the homomorphic extension of θ to t.

For an equational theory E, the *satisfaction relation* $(tr, \theta) \vDash_E \varphi$ between traces tr, valuations θ, and trace formulas φ is defined as follows.

$$\begin{array}{ll} (tr,\theta) \vDash_E f@i & \text{iff } \theta(i) \in \mathit{idx}(tr) \text{ and } f\theta \in_E tr_{\theta(i)} \\ (tr,\theta) \vDash_E i \lessdot j & \text{iff } \theta(i) < \theta(j) \\ (tr,\theta) \vDash_E i \doteq j & \text{iff } \theta(i) = \theta(j) \\ (tr,\theta) \vDash_E t_1 \approx t_2 & \text{iff } t_1\theta =_E t_2\theta \\ (tr,\theta) \vDash_E \neg\varphi & \text{iff not } (tr,\theta) \vDash_E \varphi \\ (tr,\theta) \vDash_E \varphi \wedge \psi & \text{iff } (tr,\theta) \vDash_E \varphi \text{ and } (tr,\theta) \vDash_E \psi \\ (tr,\theta) \vDash_E \exists x{:}s.\,\varphi & \text{iff there is } u \in \mathbf{D}_s \text{ such that} \\ & \qquad (tr, \theta[x \mapsto u]) \vDash_E \varphi \end{array}$$

The semantics of the remaining logical connectives and quantifiers are defined by translation to the given fragment as usual. Overloading notation, we write $tr \vDash_E \varphi$ if $(tr,\theta) \vDash_E \varphi$ for all θ. For a set of traces TR, we write $\mathit{TR} \vDash_E \varphi$ if $tr \vDash_E \varphi$ for all $tr \in \mathit{TR}$. We say that a *protocol P satisfies φ*, written $P \vDash_{E_{DH}} \varphi$, if $\mathit{traces}(P) \vDash_{E_{DH}} \varphi$.

C. Example: Security of NAXOS in the eCK Model

We formalize the NAXOS protocol for the eCK model using the rules in Figure 3. We include two free function symbols h_1 and h_2 in Σ_{DH}. The first rule models the generation and registration of long-term asymmetric keys. An exponent lk_A is randomly chosen and stored as the long-term key of an agent A. The persistent facts $!\mathsf{Ltk}(A, lk_A)$ and $!\mathsf{Pk}(A, \mathsf{g}\,\hat{}\,lk_A)$ denote the association between A and his long-term private and public keys. The public key is additionally sent to the adversary.

In the rules modeling the initiator and responder, each protocol thread chooses a unique ephemeral key esk_x, which we also use to identify the thread. The first initiator rule chooses the actor I and the intended partner R, looks up I's long-term key, and sends the half-key hk_I. The fact $\mathsf{Init}_1(esk_I, I, R, lk_I, hk_I)$ then stores the state of thread esk_I and the fact $!\mathsf{Ephk}(esk_I, esk_I)$ is added to allow the adversary to reveal the ephemeral key esk_I (the second argument) of the thread esk_I (the first argument). The second initiator rule reads the thread's state, looks up the public key of the intended partner, and receives the half-key Y. The key k_I is then computed. The action $\mathsf{Accept}(esk_I, I, R, k_I)$ denotes that the thread esk_I finished with the given parameters.

To specify when two threads are intended communication partners ("matching sessions"), we include $\mathsf{Sid}(esk_I, sid)$ and $\mathsf{Match}(esk_I, sid')$ actions. A thread s' matches a thread s if there exists a sid such that $\mathsf{Sid}(s', sid)$ and $\mathsf{Match}(s, sid)$ occur in the trace. By appropriately defining the Match

Generate long-term keypair:

$\mathsf{Fr}(lk_A) \text{-}[\,]\!\!\rightarrow \mathsf{!Ltk}(A{:}pub, lk_A), \mathsf{!Pk}(A, \mathsf{g}\,\hat{}\,lk_A), \mathsf{Out}(\mathsf{g}\,\hat{}\,lk_A)$

Initiator step 1:

$\mathsf{Fr}(esk_I), \mathsf{!Ltk}(I, lk_I)$

$\quad \text{-}[\,]\!\!\rightarrow \mathsf{Init}_1(esk_I, I, R{:}pub, lk_I, hk_I), \mathsf{!Ephk}(esk_I, esk_I), \mathsf{Out}(hk_I)$

where $hk_I = \mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_I, lk_I)$

Initiator step 2:

$\mathsf{Init}_1(esk_I, I, R, lk_I, hk_I), \mathsf{!Pk}(R, pk_R), \mathsf{In}(Y)$

$\quad \text{-}[\ \mathsf{Accept}(esk_I, I, R, k_I), \mathsf{Sid}(esk_I, \langle \mathsf{Init}, I, R, hk_I, Y\rangle)$

$\quad\ , \mathsf{Match}(esk_I, \langle \mathsf{Resp}, R, I, hk_I, Y\rangle)\ ]\!\!\rightarrow \mathsf{!Sessk}(esk_I, k_I)$

where $k_I = \mathsf{h}_2(Y\,\hat{}\,lk_I, pk_R\,\hat{}\,\mathsf{h}_1(esk_I, lk_I), Y\,\hat{}\,\mathsf{h}_1(esk_I, lk_I), I, R)$

Responder:

$\mathsf{Fr}(esk_R), \mathsf{!Ltk}(R, lk_R), \mathsf{!Pk}(I, pk_I), \mathsf{In}(X)$

$\quad \text{-}[\ \mathsf{Accept}(esk_R, R, I, k_R), \mathsf{Sid}(esk_R, \langle \mathsf{Resp}, R, I, X, hk_R\rangle)$

$\quad\ , \mathsf{Match}(esk_R, \langle \mathsf{Init}, I, R, X, hk_R\rangle)\ ]\!\!\rightarrow$

$\mathsf{!Sessk}(esk_R, k_R), \mathsf{!Ephk}(esk_R, esk_R), \mathsf{Out}(\mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_R, lk_R))$

where $hk_R = \mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_R, lk_R)$, and

$k_R = \mathsf{h}_2(pk_I\,\hat{}\,\mathsf{h}_1(esk_R, lk_R), X\,\hat{}\,lk_R, X\,\hat{}\,\mathsf{h}_1(esk_R, lk_R), I, R)$

Key Reveals for the eCK model:

$\mathsf{!Sessk}(tid, k)$	$\text{-}[\ \mathsf{SesskRev}(tid)\ ]\!\!\rightarrow$	$\mathsf{Out}(k)$
$\mathsf{!Ltk}(A, lk_A)$	$\text{-}[\ \mathsf{LtkRev}(A)\ ]\!\!\rightarrow$	$\mathsf{Out}(lk_A)$
$\mathsf{!Ephk}(tid, esk_A)$	$\text{-}[\ \mathsf{EphkRev}(tid)\ ]\!\!\rightarrow$	$\mathsf{Out}(esk_A)$

Figure 3. Multiset rewriting rules formalizing NAXOS.

actions and session identifier sid, various definitions of matching can be modeled [21].

Finally, the fact $\mathsf{!Sessk}(esk_I, k_I)$ is added to the second initiator rule to allow revealing the session key k_I. The responder rule works analogously. The final three rules model that, in the eCK model, the adversary can reveal any session, long-term, or ephemeral key. We model the restrictions on key reveals as part of the security property and thus record all key reveals in the trace.

We formalize security in the eCK model by the formula in Figure 4, which is a one-to-one mapping of the original definition of eCK security given in [2]. Intuitively, the formula states that if the adversary knows the session key of a thread esk_I, then he must have performed forbidden key reveals. The left-hand side of the implication states that the key k is known and the right-hand side disjunction states the restrictions on key reveals. We describe each disjunct in the comment above it. Further motivation and variants of these restrictions can be found in [2], [21]. Note that the eCK model formalizes weak Perfect Forward Secrecy (weak PFS), as it only allows for a long-term key reveal of the intended partner if there is a matching session. To obtain a variant of eCK formalizing PFS, we can replace the last line with

$$\vee\ (\exists i_5.\ \mathsf{LtkRev}(B)@i_5 \wedge i_5 \lessdot i_1))\Big)\ \Big).$$

This allows the adversary to reveal the long-term key of the intended partner after the test thread is finished or if there is

$\forall i_1\, i_2\, s\, A\, B\, k.\ (\mathsf{Accept}(s, A, B, k)@i_1 \wedge \mathsf{K}(k)@i_2) \Rightarrow$

// If the session key of the test thread s is known, then
// s must be "not clean". Hence either there is a
// session key reveal for s,

$\Big((\exists i_3.\ \mathsf{SesskRev}(s)@i_3)$

// or a session key reveal for a matching session,

$\vee\ (\exists s' i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)$

$\quad \wedge (\exists i_5.\ \mathsf{SesskRev}(s')@i_5))$

// or if a matching session exists,

$\vee\ \Big(\exists s' i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)$

// both lk_A and esk_A, or both lk_B and esk_B are revealed,

$\wedge \Big((\exists i_5\, i_6.\ \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6)$

$\quad \vee (\exists i_5\, i_6.\ \mathsf{LtkRev}(B)@i_5 \wedge \mathsf{EphkRev}(s')@i_6))\Big)$

// or if no matching session exists,

$\vee\ \Big(\neg(\exists s' i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4))$

// either both lk_A and esk_A, or lk_B are revealed.

$\wedge \Big((\exists i_5\, i_6.\ \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6)$

$\quad \vee (\exists i_5.\ \mathsf{LtkRev}(B)@i_5))\Big)\ \Big)$

Figure 4. eCK security definition.

a matching session. The NAXOS protocol does not satisfy this property, as reported in Table I in SectionVII.

Note that some protocols require modeling inequality conditions, e.g., the TS1-2004 [22] protocol, which assumes that an agent never executes a session with himself. We model inequality conditions in two steps. First, we include $\mathsf{Neq}(s, t)$ facts in the actions of rules that require the terms s and t to be unequal. Second, we replace the considered security property φ with $(\neg(\exists i\, x.\ \mathsf{Neq}(x, x)@i)) \Rightarrow \varphi$ to restrict the analysis to traces where all inequality conditions hold. This filtering construction also works for enforcing other restrictions on traces, e.g., the uniqueness of certain actions.

V. Normal Dependency Graphs

For symbolic attack-search algorithms, there are several drawbacks to the multiset rewriting semantics given in the previous section. First, incrementally constructing attacks is difficult with (action-)traces, as they contain neither the history of past states nor the causal dependencies between steps. Second, symbolic reasoning modulo E_{DH} is difficult because E_{DH} contains cancellation equations. For example, if the adversary knows $t = na * x$ for a nonce na, we cannot conclude that na has been used in the construction of t, as x could be equal to na^{-1}. Third, the message deduction rules allow for redundant steps such as first encrypting a cleartext and then decrypting the resulting ciphertext. For search algorithms, it is useful to impose normal-form conditions on message deduction to avoid exploring such redundant steps.

We take the following approach. First, we define dependency graphs. They consist of the sequence of rewriting rule instances corresponding to a protocol execution and their causal dependencies, similar to strand spaces [23]. Afterwards, we show that we can use dependency graphs modulo AC, an equational theory without cancellation equations, instead of dependency graphs modulo E_{DH}. Finally, we define normal message deductions and the corresponding normal dependency graphs. We also show that normal dependency graphs are weakly trace equivalent to the multiset rewriting semantics.

A. Dependency Graphs

We use dependency graphs to represent protocol executions together with their causal dependencies. A dependency graph consists of nodes labeled with rule instances and dependencies between the nodes. We first present an example of a dependency graph and then give its formal definition.

Example 1 (Dependency Graph)**.** Consider the protocol

$$
\begin{aligned}
P = \{\ & [\mathsf{Fr}(x), \mathsf{Fr}(k)]\!-\![\,]\!\!\rightarrow\![\mathsf{St}(x,k), \mathsf{Out}(\mathrm{enc}(x,k)), \mathsf{Key}(k)] \\
, & \ [\mathsf{St}(x,k), \mathsf{In}(\langle x,x\rangle)]\!-\![\ \mathsf{Fin}(x,k)\ ]\!\!\rightarrow\![\,] \\
, & \ [\mathsf{Key}(k)]\!-\![\ \mathsf{Rev}(k)\ ]\!\!\rightarrow\![\mathsf{Out}(k)]\ \}\ .
\end{aligned}
$$

Figure 5 shows a dependency graph for an execution of P. We use inference rule notation with the actions on the right for rule instances. Nodes 1 and 2 are rule instances that create fresh names. Node 3 is an instance of the first protocol rule. Node 4 is an instance of the key reveal rule. Nodes 5–9 are instances of message deduction rules and denote that the adversary receives a ciphertext and its key, decrypts the ciphertext, pairs the resulting cleartext with itself, and sends the result to an instance of the second protocol rule, Node 10. The edges denote causal dependencies: an edge from a conclusion of node i to a premise of node j denotes that the corresponding fact is generated by i and consumed by j. Since this is a dependency graph modulo E_{DH}, it is sufficient that each pair of generated and consumed facts is equal modulo E_{DH}.

Formally, let E be an equational theory and R be a set of multiset rewriting rules. We say that $dg = (I, D)$ is a *dependency graph modulo E for R* if $I \in (\mathit{ginsts}(R \cup \{\textsc{Fresh}\}))^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions **DG1–4** listed below. To state these conditions, we introduce the following definitions. We call $\mathit{idx}(I)$ the *nodes* and D the *edges* of dg. We write $(i,u) \rightarrowtail (j,v)$ for the edge $((i,u),(j,v))$. Let $I = [l_1\!-\![\ a_1\ ]\!\!\rightarrow\! r_1, \ldots, l_n\!-\![\ a_n\ ]\!\!\rightarrow\! r_n]$. The *trace* of dg is $\mathit{trace}(dg) = [\mathit{set}(a_1), \ldots, \mathit{set}(a_n)]$. A *conclusion* of dg is a pair (i,u) such that i is a node of dg and $u \in \mathit{idx}(r_i)$. The corresponding *conclusion fact* is $(r_i)_u$. A *premise* of dg is a pair (i,u) such that i is a node of dg and $u \in \mathit{idx}(l_i)$. The corresponding *premise fact* is $(l_i)_u$. A conclusion or premise is *linear* if its fact is linear.

DG1 For every edge $(i,u) \rightarrowtail (j,v) \in D$, it holds that $i < j$ and the conclusion fact of (i,u) is equal modulo E to the premise fact of (j,v).

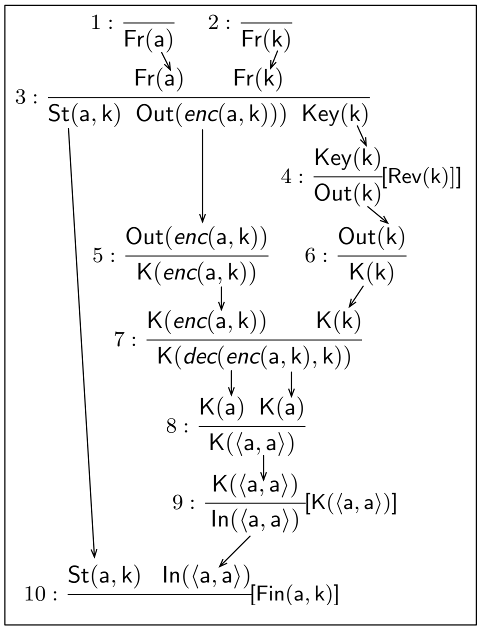

1 : Fr(a)
2 : Fr(k)
3 : Fr(a) Fr(k) / St(a, k) Out(enc(a, k))) Key(k)
4 : Key(k) / Out(k) [Rev(k)]]
5 : Out(enc(a, k)) / K(enc(a, k))
6 : Out(k) / K(k)
7 : K(enc(a, k)) K(k) / K(dec(enc(a, k), k))
8 : K(a) K(a) / K(⟨a, a⟩)
9 : K(⟨a, a⟩) / In(⟨a, a⟩) [K(⟨a, a⟩)]
10 : St(a, k) In(⟨a, a⟩) / [Fin(a, k)]

Figure 5. Dependency graph modulo E_{DH}.

DG2 Every premise of dg has exactly one incoming edge.

DG3 Every linear conclusion of dg has at most one outgoing edge.

DG4 The $\textsc{Fresh}$ rule instances in I are unique.

We denote the set of all dependency graphs modulo E for R by $\mathit{dgraphs}_E(R)$.

Note that, for all protocols P, the multiset rewriting semantics given in Section IV and the dependency graphs modulo E_{DH} for $P \cup MD$ have the same set of traces, i.e., $\mathit{traces}(P) =_{E_{DH}} \{\mathit{trace}(dg) \mid dg \in \mathit{dgraphs}_{E_{DH}}(P \cup MD)\}$.

B. Dependency Graphs modulo AC

We now switch to a semantics based on dependency graphs modulo AC. We use standard notions from order-sorted rewriting [24] and proceed in two steps.

First, we define AC as the equational theory generated by Equations (4–5) from Figure 2 and DH as the rewriting system obtained by orienting Equations (1–3,9–10) from Figure 2 and all equations from Figure 6 from left to right. $DH \uplus AC$ is an equational presentation of E_{DH} and DH is AC-convergent and AC-coherent. We can therefore define $t{\downarrow_{DH}}$ as the normal form of t with respect to DH,AC-rewriting and have $t =_{E_{DH}} s$ iff $t{\downarrow_{DH}} =_{AC} s{\downarrow_{DH}}$. We say that t is ${\downarrow_{DH}}$-normal if $t =_{AC} t{\downarrow_{DH}}$. We say a dependency graph $dg = (I, D)$ is ${\downarrow_{DH}}$-normal if all rule instances in I are ${\downarrow_{DH}}$-normal.

Second, E_{DH} has the finite variant property [25] for this presentation, which allows us to perform symbolic reasoning about normalization. More precisely, for all terms t, there is a finite set of substitutions $\{\tau_1, \ldots, \tau_k\}$ such that for all substitutions σ, there is an $i \in \{1, \ldots, k\}$ and a substitution σ' with $(t\sigma){\downarrow_{DH}} =_{AC} ((t\tau_i){\downarrow_{DH}})\sigma'$ and $(x\sigma){\downarrow_{DH}} =_{AC} x\tau_i\sigma'$

$$\begin{array}{llll}
(1) & (x^{-1}*y)^{-1} \simeq x*y^{-1} & (6) & 1^{-1} \simeq 1 \\
(2) & x^{-1}*y^{-1} \simeq (x*y)^{-1} & (7) & x*1 \simeq x \\
(3) & x*(x*y)^{-1} \simeq y^{-1} & (8) & (x^{-1})^{-1} \simeq x \\
(4) & x^{-1}*(y^{-1}*z) \simeq (x*y)^{-1}*z & (9) & x*(x^{-1}*y) \simeq y \\
(5) & (x*y)^{-1}*(y*z) \simeq x^{-1}*z & (10) & x*x^{-1} \simeq 1
\end{array}$$

Figure 6. Lankford's presentation of the abelian group axioms

for all $x \in \mathit{vars}(t)$. We call $\{(t\tau_i{\downarrow_{DH}}, \tau_i) \mid 1 \leq i \leq k\}$ a *complete set of DH,AC-variants of t*. For a given term t, we use folding variant narrowing [24] to compute such a set, which we denote by $\lceil t \rceil^{DH}$. Overloading notation, we also denote $\{s \mid (s,\tau) \in \lceil t \rceil^{DH}\}$ by $\lceil t \rceil^{DH}$.

It is straightforward to extend these notions to multiset rewriting rules by considering rules as terms and the required new function symbols as free. We can then show that $\mathit{dgraphs}_{E_{DH}}(R){\downarrow_{DH}} \subseteq_{AC} \mathit{dgraphs}_{AC}(\lceil R \rceil^{DH})$ for all sets of multiset rewriting rules R. If we restrict the right hand side to $\downarrow_{DH}$-normal dependency graphs, then the two sets are equal modulo AC.

Example 2. To normalize the graph $dg = (I, D)$ in Figure 5 with respect to $\downarrow_{DH}$, it suffices to replace I_7 with $ri = \mathsf{K}(\mathsf{enc}(\mathsf{a},\mathsf{k})), \mathsf{K}(\mathsf{k}) \text{-}[]\!\!\rightarrow \mathsf{K}(\mathsf{a})$, calling the result dg'. Since ri is not an instance of the decryption rule $\mathit{rdec} = \mathsf{K}(x), \mathsf{K}(y) \text{-}[]\!\!\rightarrow \mathsf{K}(\mathsf{dec}(x,y))$ or any other rule in $P \cup \mathit{MD}$, dg' is not in $\mathit{dgraphs}_{AC}(P \cup \mathit{MD})$. However, ri is an instance of $\mathsf{K}(\mathsf{enc}(x,y)), \mathsf{K}(y) \text{-}[]\!\!\rightarrow \mathsf{K}(x)$, which is a DH,AC-variant of rdec and therefore in $\lceil \mathit{MD} \rceil^{DH}$. Hence, $dg' \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH})$.

C. Normal Dependency Graphs

We first define the class of $*$-restricted protocols, which do not multiply exponents. We then define rules for normal message deduction and the corresponding normal dependency graphs, which are weakly trace equivalent to the multiset rewriting semantics for $*$-restricted protocols.

1) $*$-restricted Protocols: The following restriction ensures that protocols do not multiply exponents and do not introduce products by other means. A protocol P is *$*$-restricted* if, for each of its rules $l \text{-}[\, a \,]\!\!\rightarrow r$, (a) l does not contain the function symbols $*$, $\hat{}$, ${}^{-1}$, fst, snd, and dec, and (b) r does not contain the function symbol $*$.

In general, condition (a) prevents protocol rules from pattern matching on reducible function symbols. Condition (b) prevents protocols from directly using multiplication, although repeated exponentiation is still allowed. Note that these restrictions are similar to those of previous work such as [7], [8] and are not a restriction in practice. Protocols that use multiplication in the group of exponents can usually be specified by using repeated exponentiation. Moreover, protocols that use multiplication in the DH group, such as MQV [26], cannot be specified anyway since $*$ denotes multiplication in the group of exponents.

For $*$-restricted protocols, products that occur in positions that can be extracted by the adversary can always be constructed by the adversary himself from their components.

2) Normal Message Deduction: Message deduction steps in dependency graphs modulo AC use rules from $\lceil \mathit{MD} \rceil^{DH}$. These rules still allow redundant steps. We now eliminate some of them by tagging the rules to limit their applicability.

We first partition $\lceil \mathit{MD} \rceil^{DH}$ into five subsets: communication rules for sending and receiving messages, multiplication rules consisting of all DH,AC-variants of the rule for multiplication, construction rules that apply a function symbol to arguments, deconstruction rules that extract a subterm from an argument, and the remaining exponentiation rules, which are all DH,AC-variants of the rule for exponentiation and are neither construction nor deconstruction rules.

We use tags to forbid two types of redundancies. First, we forbid using a deconstruction rule to deconstruct the result of a construction rule. This is analogous to restrictions for normal natural deduction proofs, adapted to the setting of message deduction [27]. Second, we forbid repeated exponentiation, which can always be replaced by a single exponentiation with the product of all exponents. In particular, we use the *deconstruction tags* $\downarrow$ and $\uparrow$ and the *exponentiation tags* exp and noexp. We use $\downarrow$ to tag K-facts where deconstruction is allowed and $\uparrow$ where deconstruction is forbidden. We use exp to tag K-facts that can be used as the base of an exponentiation and noexp where this is forbidden.

We obtain the *normal message deduction rules ND* shown in Figure 9 as follows. First, we add the COERCE rule to switch from message deconstruction to message construction, preserving the exponentiation tag. Second, we replace the multiplication rules by l-ary construction rules for multiplication. For $*$-restricted protocols, these rules are sufficient to reason about products. Third, we use exponentiation tags as follows. The construction rule for exponentiation, the deconstruction rules for exponentiation, and the exponentiation rules use exp for the first premise (the base), a variable for the second premise, and noexp for the conclusion. The remaining rules use variables for the premises and exp for the conclusion. Finally, we use deconstruction tags as follows. We use $\uparrow$ for the conclusion of construction rules and $\downarrow$ for the first premise of deconstruction and exponentiation rules. This ensures that a deconstruction or exponentiation rule can never use the conclusion of a construction rule as its first premise. For the remaining premises of rules, we use $\uparrow$. For the remaining conclusions of rules, we use $\downarrow$. Note that all conclusions of exponentiation rules, including the ones that we do not show, are of the form $\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(t \,\hat{}\, s)$ and can therefore only be used by COERCE.

Example 3. Figure 7 shows five message deduction subgraphs. In (a), the adversary decrypts a message that he earlier

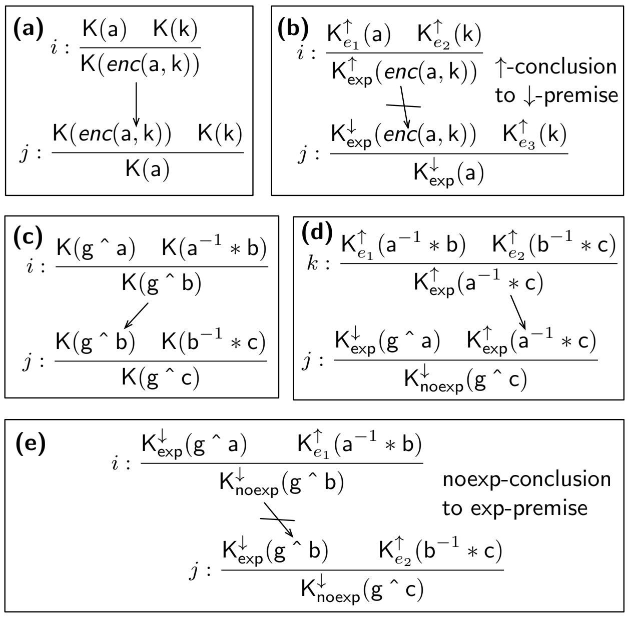


Figure 7. Message deduction subgraphs for encryption. We use $\not\twoheadrightarrow$ for edges that are invalid because the source and target are not equal. We use $i, j, k \in \mathbb{N}$ and the exponentiation tags e_1, e_2, and e_3.

encrypted himself. Instead of performing these deductions, the adversary can directly use the conclusion $\mathsf{K}(\mathsf{a})$ that is used by the encryption. The deduction from (a) is not possible with the normal message deduction rules *ND* because the $\uparrow$-tags and $\downarrow$-tags prevent applying a deconstruction rule to the conclusion of a construction rule, as depicted in (b). In (c), the adversary performs a redundant step that involves repeated exponentiation. Note that an unbounded number of steps that add a new exponent and remove the previously added exponent can be inserted inbetween the two rules. This deduction is impossible with the normal message deduction rules because a conclusion with a noexp-tag cannot be used with a premise that requires an exp-tag, as depicted in (e). We can replace the repeated exponentiation with one multiplication and one exponentiation as depicted in (d).

3) Normal Dependency Graphs: We now define normal dependency graphs. They use the normal message deduction rules and enforce further normal-form conditions. To state the conditions, we define the *input components of a term t* as $inp(t)$, such that $inp(t^{-1}) = inp(t)$, $inp(\langle t_1, t_2\rangle) = inp(t_1) \cup inp(t_2)$, $inp(t_1 * t_2) = inp(t_1) \cup inp(t_2)$, and $inp(t) = \{t\}$ otherwise. Intuitively, $inp(t)$ consists of the maximal subterms of t that are not products, pairs, or inverses.

Formally, a *normal dependency graph for a protocol P* is a dependency graph dg such that $dg \in dgraphs_{AC}(\lceil P\rceil^{DH} \cup ND)$ and the following conditions are satisfied.

N1 The dependency graph dg is $\downarrow_{DH}$-normal.
N2 No instance of COERCE deduces a pair or an inverse.
N3 There is no multiplication rule that has a premise fact of the form $\mathsf{K}^{\uparrow}_{e}(t * s)$.

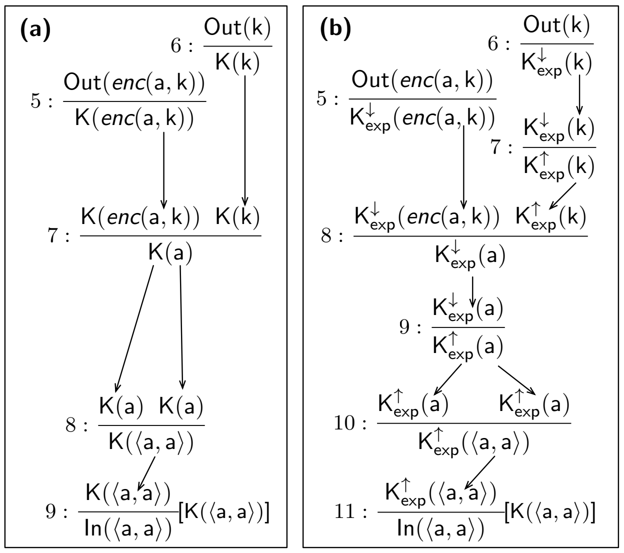


Figure 8. Examples of message deduction subgraphs of (a) a dependency graph modulo AC and (b) a normal dependency graph.

N4 All conclusion facts $\mathsf{K}^{d}_{e}(t * s)$ are conclusions of a multiplication rule.
N5 If there are two conclusions c and c' with conclusion facts $\mathsf{K}^{d}_{e}(m)$ and $\mathsf{K}^{d}_{e'}(m')$ such that $m =_{AC} m'$, then $c = c'$.
N6 If there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\downarrow}_{e}(m)$ and a conclusion $(j, 1)$ with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ such that $m =_{AC} m'$, then $i < j$ and j is an instance of COERCE or the construction rule for pairing or the one for inversion.
N7 For all nodes $\mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{e}(t_1) -[\,]\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2 \,\hat{}\, t_2)$ such that s_2 is of sort *pub*, $inp(t_2) \notin inp(t_1)$.

We denote the set of all normal dependency graphs of P by *ndgraphs*(P).

N1 ensures that all rule instances are $\downarrow_{DH}$-normal. **N2** ensures that pairs and inverses are always completely deconstructed. **N3** and **N4** formalize that the adversary constructs all products directly by multiplying their components. This does not restrict the adversary since we limit ourselves to $*$-restricted protocols. **N5** and **N6** ensure a restricted form of message uniqueness. **N7** forbids instances of exponentiation rules that can be replaced by instances of the construction rule for exponentiation where the base is a public name.

Note that normal dependency graphs allow exactly the same executions as our multiset rewriting semantics.

Lemma 1. *For all $*$-restricted protocols P,*

$$\overline{traces}(P)\downarrow_{DH} =_{AC} \{\overline{trace}(dg) \mid dg \in ndgraphs(P)\} .$$

Example 4. Figure 8 shows two message deduction subgraphs. Subfigure (a) shows the message deduction subgraph of a dependency graph modulo AC and (b) shows the message deduction subgraph of the corresponding normal dependency graph. We obtain (b) from (a) by renumbering the nodes and adding the required tags and COERCE nodes.

Coerce rule: $\textsc{Coerce}\ \dfrac{\mathsf{K}^{\downarrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{e}(x)}$ *Communication rules*: $\textsc{Irecv}\ \dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$ $\textsc{Isend}\ \dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]$

Construction rules:

$$\dfrac{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x)\ \ \mathsf{K}^{\uparrow}_{e}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(x\,\hat{}\,y)} \quad \dfrac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}pub)} \quad \dfrac{\mathsf{Fr}(x{:}fresh)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}fresh)} \quad \dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x^{-1})} \quad \dfrac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(1)} \quad \dfrac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{enc}(x,y))} \quad \dfrac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{dec}(x,y))}$$

$$\dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{h}(x))} \quad \dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{fst}(x))} \quad \dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{snd}(x))} \quad \dfrac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\langle x,y\rangle)} \quad \dfrac{\mathsf{K}^{\uparrow}_{e_1}(x_1)\ \dots\ \mathsf{K}^{\uparrow}_{e_n}(x_n) \qquad \mathsf{K}^{\uparrow}_{e_{n+1}}(x_{n+1})\ \dots\ \mathsf{K}^{\uparrow}_{e_l}(x_l)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}((x_1 * \dots * x_n) * (x_{n+1} * \dots * x_l)^{-1})}$$

Deconstruction rules:

$$\dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,y^{-1})\ \ \mathsf{K}^{\uparrow}_{e}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,(y * z^{-1}))\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)}$$

$$\dfrac{\mathsf{K}^{\downarrow}_{e}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \quad \dfrac{\mathsf{K}^{\downarrow}_{e}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(y)} \quad \dfrac{\mathsf{K}^{\downarrow}_{e}(x^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \quad \dfrac{\mathsf{K}^{\downarrow}_{e_1}(\mathrm{enc}(x,y))\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$$

Exponentiation rules:

$$\dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}_{e}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\,\hat{}\,(y * z))} \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\,\hat{}\,z)} \quad \dots \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\,\hat{}\,(y * z^{-1}))\ \ \mathsf{K}^{\uparrow}_{e}(a * b^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\,\hat{}\,(y * a * (z * b)^{-1}))}$$

Figure 9. Normal message deduction rules *ND*. Rules containing variables e or e_i denote all variants where these are replaced by noexp or exp. Rules containing n and l denote all variants for $n \geq 1$ and $l \geq 2$. There are 42 exponentiation rules computed from the DH,AC-variants of the exponentiation rule.

Example 5. Consider the exponentiation-construction node $i : \mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{g}\,\hat{}\,\mathsf{a}), \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{a}^{-1} * \mathsf{b}) \text{-}[\,]\!\rightarrowtail \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(\mathsf{g}\,\hat{}\,\mathsf{b})$. The conclusion of i either has no outgoing edge or a single edge to a COERCE node j. In the first case, the node i can be removed. In the second case, the nodes i and j can be replaced by $j : \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{g}), \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{b}) \text{-}[\,]\!\rightarrowtail \mathsf{K}^{\uparrow}_{\mathsf{noexp}}(\mathsf{g}\,\hat{}\,\mathsf{b})$, keeping all the outgoing edges of j. This replacement is possible because g is deducible and conditions **N4** and **N3** ensure that b is deducible whenever $\mathsf{a}^{-1} * \mathsf{b}$ is.

4) Properties of Normal Dependency Graphs: We prove two properties of normal dependency graphs that are crucial for our search algorithm. The first property states that every $\mathsf{K}^{\downarrow}_{e}(t)$-premise is deduced using a chain of deconstruction rules from a received message. We use here the *extended set of deconstruction rules ND^{destr}* that consists of the deconstruction and exponentiation rules from Figure 9. To define the second property, we partition the construction rules into the *implicit construction rules $ND^{c\text{-}impl}$* consisting of the pair, inversion, and multiplication construction rules and the *explicit construction rules $ND^{c\text{-}expl}$* consisting of the remaining construction rules and the COERCE rule. Since all messages that are products, pairs, and inverses must be constructed with implicit construction rules, we can show that if a $K^{\uparrow}_{e}(t)$ conclusion was deduced, then every message in $inp(t)$ must have been previously deduced.

Let $dg = (I, D)$ be a normal dependency graph for P. Its *deconstruction chain relation* $\dashrightarrow_{dg}$ is the smallest relation such that $c \dashrightarrow_{dg} p$ if c is a $\mathsf{K}^{\downarrow}$-conclusion in dg and (a) $c \rightarrowtail p \in D$ or (b) there is a premise (j, u) such that $c \rightarrowtail (j, u) \in D$ and $(j, 1) \dashrightarrow_{dg} p$. Our search algorithm exploits the following lemma to reason about the possible origins of $\mathsf{K}^{\downarrow}$-premises.

Lemma 2 (Deconstruction Chain)**.** *For every premise p with fact $\mathsf{K}^{\downarrow}_{e}(t)$ of dg, there is a node i in dg such that $I_i \in ginsts(\textsc{Irecv})$ and $(i, 1) \dashrightarrow_{dg} p$.*

The *implicit construction dependency relation $\twoheadrightarrow_{dg}$ of dg* is the smallest relation such that $c \twoheadrightarrow_{dg} p$ if there is a premise (j, u) with $I_j \in ginsts(ND^{c\text{-}impl})$ such that (a) $c \rightarrowtail (j, u) \in D$ and $(j, 1) \rightarrowtail p \in D$ or (b) $c \twoheadrightarrow_{dg} (j, u)$ and $(j, 1) \rightarrowtail p \in D$. Our algorithm uses the following lemma to keep the construction of pairs, inverses, and products implicit in the search.

Lemma 3 (Implicit Construction)**.** *For every premise p in dg with fact $\mathsf{K}^{\uparrow}_{e}(t)$ and every message $m \in_{AC} inp(t)$ with $m \neq_{AC} t$, there is a conclusion $(i, 1)$ in dg with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ such that $I_i \in ginsts(ND^{c\text{-}expl})$, $m' =_{AC} m$, and $(i, 1) \twoheadrightarrow_{dg} p$.*

Example 6. For the normal message deduction subgraph dg in Figure 8b, $(5, 1) \dashrightarrow_{dg} (9, 1)$, $(8, 1) \dashrightarrow_{dg} (9, 1)$, and $(6, 1) \dashrightarrow_{dg} (7, 1)$, but not $(6, 1) \dashrightarrow_{dg} (9, 1)$ because $(8, 2)$ is not a $\mathsf{K}^{\downarrow}$-premise. We have $(9, 1) \twoheadrightarrow_{dg} (11, 1)$, but not $(10, 1) \twoheadrightarrow_{dg} (11, 1)$ because $\twoheadrightarrow_{dg}$ requires at least one *inner* implicit construction node.

VI. AUTOMATED PROTOCOL ANALYSIS

In this section, we give an algorithm for determining whether $P \vDash_{E_{DH}} \varphi$ for a $*$-restricted protocol P and a guarded trace property φ. Guarded trace properties are an

Trace formula reduction rules:

$\mathbf{S}_{\approx}$:	$\Gamma \rightsquigarrow_P \Vert_{\sigma \in \mathit{unify}_{AC}(t_1,t_2)}(\Gamma\sigma)$	if $(t_1 \approx t_2) \in \Gamma$ and $t_1 \neq_{AC} t_2$
$\mathbf{S}_{\doteq}$:	$\Gamma \rightsquigarrow_P \Gamma\{i/j\}$	if $(i \doteq j) \in \Gamma$ and $i \neq j$
$\mathbf{S}_{@}$:	$\Gamma \rightsquigarrow_P \Vert_{ri \in \lceil P \rceil^{DH} \cup \{\textsc{Isend}\}} \Vert_{f' \in \mathit{acts}(ri)}(i : ri,\, f \approx f',\, \Gamma)$	if $(f@i) \in \Gamma$ and $(f@i) \notin_{AC} \mathit{as}(\Gamma)$
$\mathbf{S}_{\bot}$:	$\Gamma \rightsquigarrow_P \bot$	if $\bot \in \Gamma$
$\mathbf{S}_{\neg,\approx}$:	$\Gamma \rightsquigarrow_P \bot$	if $\neg(t \approx t) \in_{AC} \Gamma$
$\mathbf{S}_{\neg,\doteq}$:	$\Gamma \rightsquigarrow_P \bot$	if $\neg(i \doteq i) \in \Gamma$
$\mathbf{S}_{\neg,@}$:	$\Gamma \rightsquigarrow_P \bot$	if $\neg(f@i) \in \Gamma$ and $(f@i) \in \mathit{as}(\Gamma)$
$\mathbf{S}_{\neg,\lessdot}$:	$\Gamma \rightsquigarrow_P (i \lessdot j, \Gamma) \Vert (\Gamma\{i/j\})$	if $\neg(j \lessdot i) \in \Gamma$ and neither $i \lessdot_\Gamma j$ nor $i = j$
$\mathbf{S}_{\vee}$:	$\Gamma \rightsquigarrow_P (\phi_1, \Gamma) \Vert (\phi_2, \Gamma)$	if $(\phi_1 \vee \phi_2) \in_{AC} \Gamma$ and $\{\phi_1, \phi_2\} \cap_{AC} \Gamma = \varnothing$
$\mathbf{S}_{\wedge}$:	$\Gamma \rightsquigarrow_P (\phi_1, \phi_2, \Gamma)$	if $(\phi_1 \wedge \phi_2) \in_{AC} \Gamma$ and not $\{\phi_1, \phi_2\} \subseteq_{AC} \Gamma$
$\mathbf{S}_{\exists}$:	$\Gamma \rightsquigarrow_P (\phi\{y/x\}, \Gamma)$	if $(\exists x{:}s.\, \phi) \in \Gamma$, $\phi\{w/x\} \notin_{AC} \Gamma$ for every term w of sort s, and $y{:}s$ fresh
$\mathbf{S}_{\forall}$:	$\Gamma \rightsquigarrow_P (\psi\sigma, \Gamma)$	if $(\forall \vec{x}.\, \neg(f@i) \vee \psi) \in \Gamma$, $\mathit{dom}(\sigma) = \mathit{set}(\vec{x})$, $(f@i)\sigma \in_{AC} \mathit{as}(\Gamma)$, and $\psi\sigma \notin_{AC} \Gamma$

Graph constraint reduction rules:

$\mathbf{U}_{\mathit{lbl}}$:	$\Gamma \rightsquigarrow_P (ri \approx ri', \Gamma)$	if $\{i : ri,\, i : ri'\} \subseteq \Gamma$ and $ri \neq_{AC} ri'$
$\mathbf{DG1}_1$:	$\Gamma \rightsquigarrow_P \bot$	if $i \lessdot_\Gamma i$
$\mathbf{DG1}_2$:	$\Gamma \rightsquigarrow_P (f \approx f', \Gamma)$	if $c \rightarrowtail p \in \Gamma$, $(c, f) \in \mathit{cs}(\Gamma)$, $(p, f') \in \mathit{ps}(\Gamma)$, and $f \neq_{AC} f'$
$\mathbf{DG2}_1$:	$\Gamma \rightsquigarrow_P$ (if $u = v$ then $\Gamma\{i/j\}$ else $\bot$)	if $\{(i,v) \rightarrowtail p,\, (j,u) \rightarrowtail p\} \subseteq \Gamma$ and $i \neq j$
$\mathbf{DG2}_{2,P}$:	$\Gamma \rightsquigarrow_P \Vert_{ri \in \lceil P \rceil^{DH} \cup \{\textsc{Isend},\textsc{Fresh}\}} \Vert_{u \in \mathit{idx}(\mathit{concs}(ri))}(i : ri,\, (i,u) \rightarrowtail p,\, \Gamma)$	if p is an open f-premise in Γ, f is not a $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-fact, and i fresh
$\mathbf{DG3}$:	$\Gamma \rightsquigarrow_P$ (if $u = v$ then $\Gamma\{i/j\}$ else $\bot$)	if $\{c \rightarrowtail (i,v), c \rightarrowtail (j,u)\} \subseteq \Gamma$, c linear in Γ, and $i \neq j$,
$\mathbf{DG4}$:	$\Gamma \rightsquigarrow_P \Gamma\{i/j\}$	if $\{i : \text{-}[\,]\!\!\rightarrow \mathsf{Fr}(m),\, j : \text{-}[\,]\!\!\rightarrow \mathsf{Fr}(m)\} \subseteq_{AC} \Gamma$ and $i \neq j$
$\mathbf{N1}$:	$\Gamma \rightsquigarrow_P \bot$	if $(i : ri) \in \Gamma$ and ri not $\downarrow_{DH}$-normal
$\mathbf{N5,6}$:	$\Gamma \rightsquigarrow_P \Gamma\{i/j\}$	if $\{((i,1), \mathsf{K}^{d}_{e}(t)),\, ((j,1), \mathsf{K}^{d'}_{e'}(t))\} \subseteq_{AC} \mathit{cs}(\Gamma)$, $i \neq j$, *and* $d = d'$ or $\{i,j\} \cap \{k \mid \exists ri \in \mathit{insts}(\{\textsc{pair}\!\uparrow, \textsc{inv}\!\uparrow, \textsc{Coerce}\}).\, (k : ri) \in \Gamma\} = \varnothing$
$\mathbf{N6}$:	$\Gamma \rightsquigarrow_P (i \lessdot j, \Gamma)$	if $((j,v), \mathsf{K}^{\uparrow}_{e'}(t)) \in \mathit{ps}(\Gamma)$, $m \in_{AC} \mathit{inp}(t)$, $((i,u), \mathsf{K}^{\downarrow}_{e}(m)) \in \mathit{cs}(\Gamma)$, and not $i \lessdot_\Gamma j$
$\mathbf{N7}$:	$\Gamma \rightsquigarrow_P \bot$	if $(i : \mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{e}(t_1) \text{-}[\,]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2 \,\hat{}\, t_2)) \in \Gamma$, s_2 is of sort pub, and $\mathit{inp}(t_2) \subseteq \mathit{inp}(t_1)$

Message deduction constraint reduction rules:

$\mathbf{DG2}_{2,\uparrow i}$:	$\Gamma \rightsquigarrow_P \Vert_{(l \text{-}[\,]\!\!\rightarrow \mathsf{K}^{\uparrow}_{e}(t)) \in \mathit{ND}^{\mathit{c\text{-}expl}}}(i : (l \text{-}[\,]\!\!\rightarrow \mathsf{K}^{\uparrow}_{e}(t)),\, t \approx m,\, (i,1) \twoheadrightarrow p,\, \Gamma)$	if p is an open implicit m-construction in Γ, m non-trivial, and i fresh
$\mathbf{DG2}_{2,\uparrow e}$:	$\Gamma \rightsquigarrow_P \Vert_{ri \in \mathit{ND}^{\mathit{c\text{-}expl}}}(i : ri,\, (i,1) \rightarrowtail p,\, \Gamma)$	if p is an open $\mathsf{K}^{\uparrow}_{e}(m)$-premise in Γ, $\{m\} = \mathit{inp}(m)$, m non-trivial, and i fresh
$\mathbf{DG2}_{2,\downarrow}$:	$\Gamma \rightsquigarrow_P (i : \mathsf{Out}(y) \text{-}[\,]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{exp}}(y),\, (i,1) \dashrightarrow p,\, \Gamma)$	if p is an open $\mathsf{K}^{\downarrow}_{e}(m)$-premise in Γ and y, i fresh
$\mathbf{DG2}_{\dashrightarrow}$:	$(c \dashrightarrow p, \Gamma) \rightsquigarrow_P (c \rightarrowtail p, \Gamma) \Vert \Vert_{ri \in \mathit{ND}^{\mathit{destr}}}(i : ri,\, c \rightarrowtail (i,1),\, (i,1) \dashrightarrow p,\, \Gamma)$	if $(c, \mathsf{K}^{\downarrow}_{e}(m)) \in \mathit{cs}(\Gamma)$, $m \notin \mathcal{V}_{\mathit{msg}}$, and i fresh

We assume that the multiset rewriting rules in $\lceil P \rceil^{DH}$, $\mathit{ND}^{\mathit{c\text{-}expl}}$, and $\mathit{ND}^{\mathit{destr}}$ are renamed apart from Γ. We write $\Gamma\{a/b\}$ for the substitution of all occurrences of b with a in Γ. We write $\Gamma \rightsquigarrow_P \Gamma_1 \Vert \ldots \Vert \Gamma_n$ for $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, which denotes an n-fold case distinction. We overload notation and write $\bot$ for the empty set of constraint systems.

Figure 10. Rules defining the constraint-reduction relation $\rightsquigarrow_P$, explained in Sections VI-C and VI-D.

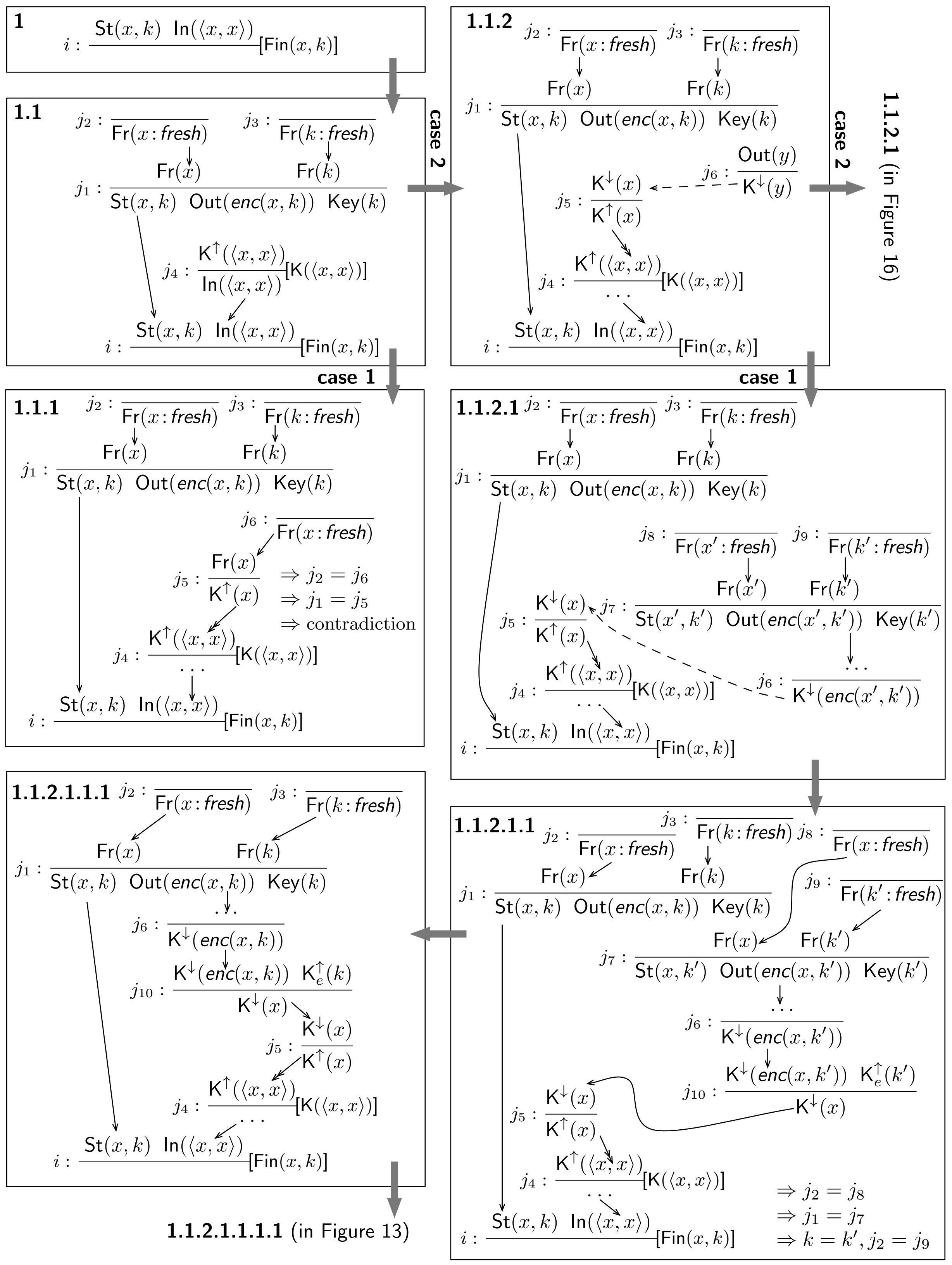


Figure 11. Constraint systems constructed by our algorithm, as explained in Example 7, when verifying $\forall x\, k\, i.\mathsf{Fin}(x,k)@i \Rightarrow \exists j.\mathsf{Rev}(k)@j$ for the protocol from Example 1. The large gray arrows denote constraint-reduction steps and "…" at either end of an edge refers to the fact at the other end. In every constraint system, variables with the same name are of the same sort and variables that have no sort annotation are of sort *msg*.

expressive subset of trace formulas. Our algorithm uses constraint solving to perform a complete search for counter-examples to $P \vDash_{E_{DH}} \varphi$, i.e., it attempts a proof by contradiction. This problem is undecidable and our algorithm does not always terminate. Nevertheless, it often finds a counter-example (an attack) or succeeds in unbounded verification.

In the following, we define guarded trace properties and constraints. Afterwards we give our constraint-solving algorithm and several examples.

A. Guarded Trace Properties

In the remainder of this section, let f range over facts and i, j over temporal variables. A trace formula φ is in *negation normal form* if it is built such that negation is only applied to trace atoms and all other logical connectives are $\wedge$, $\vee$, $\forall$, or $\exists$. Such a trace formula φ is a *guarded trace formula* if all its quantifiers are of the form $\exists \vec{x}.(f@i) \wedge \psi$ or $\forall \vec{x}.\neg(f@i) \vee \psi$ for an action $f@i$, a guarded trace formula ψ, and $x \in \mathit{vars}(f@i) \cap (\mathcal{V}_{msg} \cup \mathcal{V}_{temp})$ for every $x \in \vec{x}$. A guarded trace formula φ is a *guarded trace property* if it is closed and $t \in \mathcal{V} \cup \mathit{PN}$ holds for all terms t occurring in φ.

Note that we restrict both universal and existential quantification and, as a result, the set of guarded trace properties is closed under negation. This, together with the support for quantifier alternations and the explicit comparison of timepoints, makes guarded trace properties well-suited for specifying advanced security properties. In our case studies, it was possible to automatically convert the specified security properties, including the eCK model from Figure 4, to guarded trace properties. The conversion first rewrites the given formula to negation normal form and pushes quantifiers inward. Then, it replaces each body φ of a universal quantifier that is not a disjunction with $\varphi \vee \bot$. The rewriting for existential quantifiers is analogous.

All terms in a guarded trace property must be either variables or public names. This is not a limitation in practice since the terms required to express a security property can be added to the actions of a protocol's rewriting rules. Together with the requirement of guarding all quantified variables, this ensures that guarded trace properties are invariant under $\downarrow_{DH}$-normalization of traces. Combined with Lemma 1, this allows us to switch from verification in a multiset rewriting semantics modulo E_{DH} to verification in a dependency graph semantics modulo AC.

Theorem 1. *For every $*$-restricted protocol P and every guarded trace property φ,*

$$P \vDash_{E_{DH}} \varphi \quad \textit{iff} \quad \{\mathit{trace}(dg) \mid dg \in \mathit{ndgraphs}(P)\} \vDash_{AC} \varphi \; .$$

B. Syntax and Semantics of Constraints

In the remainder of this section, let ri range over multiset rewriting rule instances, u and v over natural numbers, and φ over guarded trace formulas. A *graph constraint* is either a *node* $i : ri$, an *edge* $(i,u) \rightarrowtail (j,v)$, a *deconstruction chain* $(i,u) \dashrightarrow (j,v)$, or an *implicit construction* $(i,u) \twoheadrightarrow (j,v)$. A *constraint* is a graph constraint or a guarded trace formula.

A *structure* is a tuple (dg,θ) of a dependency graph $dg = (I,D)$ and a valuation θ. We denote the application of the homomorphic extension of θ to a rule instance ri by $ri\,\theta$. We define when the structure (dg,θ) *satisfies* a constraint γ, written $(dg,\theta) \Vvdash \gamma$, as follows.

$$\begin{array}{ll}
(dg,\theta) \Vvdash i : ri & \text{iff } \theta(i) \in \mathit{idx}(I) \text{ and } ri\,\theta =_{AC} I_{\theta(i)} \\
(dg,\theta) \Vvdash (i,u) \rightarrowtail (j,v) & \text{iff } (\theta(i),u) \rightarrowtail (\theta(j),v) \in D \\
(dg,\theta) \Vvdash (i,u) \dashrightarrow (j,v) & \text{iff } (\theta(i),u) \dashrightarrow_{dg} (\theta(j),v) \\
(dg,\theta) \Vvdash (i,u) \twoheadrightarrow (j,v) & \text{iff } (\theta(i),u) \twoheadrightarrow_{dg} (\theta(j),v) \\
(dg,\theta) \Vvdash \varphi & \text{iff } (\mathit{trace}(dg),\theta) \vDash_{AC} \varphi
\end{array}$$

A *constraint system* Γ is a finite set of constraints. The structure (dg,θ) satisfies Γ, written $(dg,\theta) \Vvdash \Gamma$, if (dg,θ) satisfies each constraint in Γ. We say that (dg,θ) is a *P-model of* Γ, if dg is a normal dependency graph for P and $(dg,\theta) \Vvdash \Gamma$. A *$P$-solution* of Γ is a normal dependency graph dg for P such that there is a valuation θ with $(dg,\theta) \Vvdash \Gamma$. Note that the free variables of a constraint system are therefore existentially quantified.

C. Constraint-Solving Algorithm

Let P be a $*$-restricted protocol and φ a guarded trace property. Exploiting Theorem 1, our algorithm searches for a counter-example to $P \vDash_{E_{DH}} \varphi$ by trying to construct a P-solution to the constraint system $\{\hat{\varphi}\}$, where $\hat{\varphi}$ is $\neg\varphi$ rewritten into negation normal form. Our algorithm is based on the constraint-reduction relation $\rightsquigarrow_P$ between constraint systems and sets of constraint systems. We use sets of constraint systems to represent case distinctions.

Intuitively, $\rightsquigarrow_P$ refines constraint systems and our algorithm works by refining the initial constraint system $\{\hat{\varphi}\}$ until it either encounters a solved system or all systems contain (trivially) contradictory constraints. In the following, we first define $\rightsquigarrow_P$ and then state our algorithm. Afterwards, we give examples that explain and illustrate both the constraint-reduction rules defining $\rightsquigarrow_P$ and our algorithm.

The rules defining the *constraint-reduction relation* $\rightsquigarrow_P$ are given in Figure 10. There are two types of constraint-reduction rules: (1) *simplification rules* that remove contradictory constraint systems or refine constraint systems by simplifying constraints and (2) *case distinction rules* that refine constraint systems by adding further constraints. The design choices underlying our rules are motivated by their use in our algorithm. It requires them to be sound and complete and we must be able to extract a P-solution from every solved constraint system, i.e., every system that is irreducible with respect to $\rightsquigarrow_P$. The rule names refer to the form of guarded trace formulas that they solve or the property of normal dependency graphs that they ensure. There are no rules for ensuring the properties **N2–4**, as they are maintained as invariants by our algorithm.

The formal definition of our constraint-reduction rules in Figure 10 relies on the following additional conventions and definitions. We extend the equality $\approx$ over terms to facts and rule instances by interpreting the constructors for facts and rule instances as free function symbols. We write PAIR$\uparrow$ for the construction rule for pairs and INV$\uparrow$ for the construction rule for inverses. Moreover, for a constraint system Γ, its *actions* $as(\Gamma)$, *premises* $ps(\Gamma)$, and *conclusions* $cs(\Gamma)$ are defined as follows.

$$\begin{aligned} as(\Gamma) &= \{f@i \mid \exists r\, a.\ (i : l\text{-}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge f \in a\} \\ ps(\Gamma) &= \{((i,u), l_u) \mid \exists r\, a.\ (i : l\text{-}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge u \in idx(l)\} \\ cs(\Gamma) &= \{((i,v), r_v) \mid \exists l\, a.\ (i : l\text{-}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge v \in idx(r)\} \end{aligned}$$

A conclusion c is *linear in* Γ if there is a linear fact f such that $(c, f) \in cs(\Gamma)$. We say that p is an *open f-premise in* Γ if $(p, f) \in ps(\Gamma)$ and p has no incoming edges or deconstruction chains in Γ. We say that p is an *open implicit m-construction in* Γ if there is a premise $(p, \mathsf{K}^{\uparrow}_{e}(t))$ of Γ with $m \in inp(t) \setminus \{t\}$ and there is no implicit construction $c \twoheadrightarrow p$ in Γ that starts from a $\mathsf{K}^{\uparrow}_{e'}(m)$-conclusion. A term m is *trivial* if $m \in \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup PN \cup \{1\}$, where the term 1 denotes the unit in the group of exponents. The *temporal order of* Γ is

$$\begin{aligned} (\lessdot_\Gamma) = \{(i,j) \mid (i \lessdot j) \in \Gamma \vee \exists u\, v.\ &((i,u) \rightarrowtail (j,v)) \in \Gamma \\ \vee\ &((i,u) \twoheadrightarrow (j,v)) \in \Gamma \\ \vee\ &((i,u) \dashrightarrow (j,v)) \in \Gamma\}^+\ . \end{aligned}$$

We call $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-premises *message deduction constraints*.

We give our constraint-solving algorithm in Figure 12. It uses a set of constraint systems as its state $\mathbf{\Omega}$. It starts with the state $\{\{\hat{\varphi}\}\}$. Afterwards, in lines 4–6, it repeatedly applies constraint-reduction steps as long as the state is non-empty and does not contain a solved constraint system. To formalize the loop condition, we use $solved(\mathbf{\Omega})$ to denote the set of solved constraint systems in $\mathbf{\Omega}$. For automated protocol analysis, we use a heuristic (explained in Appendix B-D) to make the choice in line 5. Upon termination of the while-loop, the algorithm has either found a solved constraint system (an attack) or it proved that $\{\{\hat{\varphi}\}\}$ has no P-solution and therefore $P \vDash_{E_{DH}} \varphi$ holds. The following two theorems justify the correctness of our algorithm.

Theorem 2. *The constraint-reduction relation $\rightsquigarrow_P$ is sound and complete; i.e., for every $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, the set of P-solutions of Γ is equal to the union of the sets of P-solutions of all Γ_i, with $1 \le i \le n$.*

Theorem 3. *We can construct a P-solution from every solved system in the state $\mathbf{\Omega}$ of our constraint-solving algorithm.*

The correctness of Theorem 3 relies on the properties of solved constraint systems as well as invariants maintained by our constraint-solving algorithm, as explained in Appendix B-D.

```
1: function SOLVE($P \vDash_{E_{DH}} \varphi$)
2:     $\hat{\varphi} \leftarrow \neg\varphi$ rewritten into negation normal form
3:     $\mathbf{\Omega} \leftarrow \{\{\hat{\varphi}\}\}$
4:     while $\mathbf{\Omega} \neq \emptyset$ and $solved(\mathbf{\Omega}) = \emptyset$ do
5:         choose $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_k\}$ such that $\Gamma \in \mathbf{\Omega}$
6:         $\mathbf{\Omega} \leftarrow (\mathbf{\Omega} \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}$
7:     if $solved(\mathbf{\Omega}) \neq \emptyset$
8:         then return "attack(s) found: ", $solved(\mathbf{\Omega})$
9:         else  return "verification successful"
```

Figure 12. Pseudocode of our constraint solving algorithm.

which hold for $\{\{\hat{\varphi}\}\}$ and are maintained by $\rightsquigarrow_P$. We prove both of these theorems in Appendix C-E.

D. Extended Examples

We now give two examples that provide a first intuition for our constraint-reduction rules. Our emphasis is on the rules and their application, rather than the heuristics used in our algorithm. We provide individual explanations for all of our constraint-reduction rules and their interaction in Appendix B.

Example 7. Consider the protocol P from Example 1 and the formula $\forall x\ k\ i.\ \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j$, which can be rewritten to the guarded trace property $\varphi = \forall x\ k\ i.\ \neg(\mathsf{Fin}(x,k)@i) \vee (\exists j.\ \mathsf{Rev}(k)@j \wedge \neg(\bot))$. We explain how to use our constraint-reduction rules to show that $P \vDash_{E_{DH}} \varphi$, i.e., to show that $\{\exists x\ k\ i.\ \psi\}$ has no P-solutions, for $\psi = \mathsf{Fin}(x,k)@i \wedge (\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot)$.

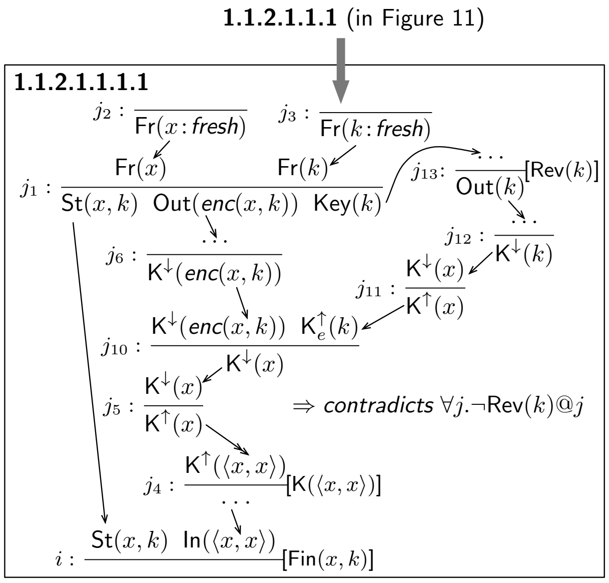
By repeated application of the rules $\mathsf{S}_\exists$ and $\mathsf{S}_\wedge$, we replace the existentially quantified variables x, k, and i with fresh free variables and split the conjunction in ψ. The resulting constraint system is

$$\begin{aligned} \{&\exists x\, k\, i.\ \psi,\ \exists k\, i.\ \psi,\ \exists i.\ \psi,\ \psi, \\ &\mathsf{Fin}(x,k)@i,\ \forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot\}\ . \end{aligned}$$

Its P-solutions are the normal dependency graphs for P that contain a node with a $\mathsf{Fin}(x,k)$ action, but no node with a $\mathsf{Rev}(k)$ action. Note that the formulas $\exists x\, k\, i.\ \psi$, $\exists k\, i.\ \psi$, $\exists i.\ \psi$, and ψ are solved in the sense that no reduction rule applies to them anymore.

The only rule applicable to the constraint system in the above state is $\mathsf{S}_@$, which enumerates all rewriting rules that could give rise to an action. We therefore apply $\mathsf{S}_@$ to the $\mathsf{Fin}(x,k)$ action of node i and solve the introduced equalities using $\mathsf{S}_\approx$. There is only one resulting constraint system, depicted in Figure 11 as System **1**. Here we interpret a constraint system as a partial, symbolic dependency graph annotated with restrictions on its trace, stated as guarded trace formulas. We do not depict the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$, as it is included in all constraint systems. We also omit the exp tags of K-facts because the protocol does not use exponentiation.

The large gray arrows in Figure 11 denote the constraint-reduction steps performed. Multiple successors result from



1.1.2.1.1.1 (in Figure 11)
1.1.2.1.1.1.1
j_2 : Fr(x: fresh)
j_3 : Fr(k: fresh)
j_1 : Fr(x) Fr(k) / St(x,k) Out(enc(x,k)) Key(k)
j_{13}: … / Out(k) [Rev(k)]
j_{12} : … / K↓(k)
j_6 : … / K↓(enc(x,k))
j_{11} : K↓(x) / K↑(x)
j_{10} : K↓(enc(x,k)) K↑e(k) / K↓(x)
j_5 : K↓(x) / K↑(x)
⇒ contradicts ∀j.¬Rev(k)@j
j_4 : K↑(⟨x,x⟩) / … [K(⟨x,x⟩)]
i : St(x,k) In(⟨x,x⟩) [Fin(x,k)]

Figure 13. Example constraint system **1.1.2.1.1.1.1**.

case distinctions. Constraint systems with no successors are contradictory for the reason given in the figure. In some cases, we contract multiple reduction steps for ease of presentation.

System **1** has multiple open non-K-premises. We solve these by repeatedly applying **DG2**$_{2,P}$ until no more open non-K-premises remain. Intuitively, rule **DG2**$_{2,P}$ solves open non-K-premises by enumerating all possible rewriting rules that have a unifying conclusion. The unification is entailed by the constraint $(i,u) \rightarrowtail p$ and the rules **DG1**$_2$ and **S**$_\approx$, which ensure the equality of facts connected by an edge. After solving all non-K-premises, the only resulting constraint system is System **1.1**.

There is only one remaining open premise in System **1.1**: $\mathsf{K}^\uparrow(\langle x,x\rangle)$. It is solved using rule **DG2**$_{2,\uparrow i}$, which exploits Lemma 3 to search directly for the possible origin of the required input component *x:fresh*. This rule improves the efficiency of our algorithm as it allows reasoning modulo pairing, inversion, and multiplication. Note that the rules **DG2**$_{2,\uparrow e}$ and **DG2**$_{2,\uparrow i}$ exclude solving $\mathsf{K}^\uparrow(x)$ premises where x is a variable of sort *msg* or *pub*, as such facts can always be constructed by the adversary. These rules may however become applicable once x is instantiated with another term in a later step. There are two multiset rewriting rules in *ND* whose conclusions are possible origins of *x:fresh*.

The case for the rule $(\mathsf{Fr}(x{:}fresh)\text{–}[]{\rightarrow}\mathsf{K}^\uparrow_{\mathsf{exp}}(x{:}fresh))$ is shown in System **1.1.1**. Note that FRESH instances must be unique and linear facts have at most one outgoing edge. This is ensured by the rules **DG4** and **DG3**, which together with the rule **U**$_{lbl}$ give rise to the chain of implications depicted in System **1.1.1**. The contradiction follows because the rewriting rule instances of nodes j_1 and j_5 are not unifiable.

The case for the COERCE rule is shown in System **1.1.2**. Note that solving $\mathsf{K}^\downarrow$-premises by enumerating the rules with unifying conclusions leads to non-termination, as the $\mathsf{K}^\downarrow$-premises of deconstruction rules are larger than their conclusions. This justifies the existence of rule **DG2**$_{2,\downarrow}$, which solves the $\mathsf{K}^\downarrow(x)$-premise by exploiting Lemma 2 to introduce a deconstruction chain starting from an IRECV instance. Rule **DG2**$_{2,P}$ then solves the open premise $\mathsf{Out}(y)$ of j_6, i.e., it enumerates all protocol rules that send messages. There are two such rules in protocol P.

System **1.1.2.1** shows the case for the first protocol rule, which is renamed apart from System **1.1.2** since node j_7 might be a different rewriting rule instance than node j_1. The deconstruction chain from j_6 to j_5 is refined by rule **DG2**$_{\twoheadrightarrow}$. This rule embodies the case distinction that a deconstruction chain is either just an edge or an edge to an instance of a rule from *ND*^{destr} and another chain starting from the conclusion of this instance. We disallow refining chains that start from a message variable, as this would lead to non-termination. Constraint systems containing such a chain are often pruned using rule **N6**, as explained in Example 8. After using rule **DG2**$_{\twoheadrightarrow}$ to refine the deconstruction chain in System **1.1.2.1** twice, we obtain System **1.1.2.1.1** where x and x' are identified and the deconstruction chain has been replaced with the edges to and from the decryption j_{10}. This is the only way to refine this chain starting from $\mathsf{K}^\downarrow(\mathrm{enc}(x',k'))$, as all other deconstruction rules lead to cases with edges between non-unifiable facts. Again the uniqueness of FRESH instances leads to a chain of implications and results in System **1.1.2.1.1.1**.

The only remaining open premise in this system is the $\mathsf{K}^\uparrow(k)$-premise of the decryption j_{10}. The constraint-reduction steps required to solve this premise are similar to the ones used to solve the $\mathsf{K}^\uparrow(x)$-premise in System **1.1**. System **1.1.2.1.1.1.1** in Figure 13 is the only one of the resulting constraint systems that is not trivially contradictory due to the uniqueness of FRESH instances. In fact, this system could be instantiated to a normal dependency graph, if it were not for the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$. Since System **1.1.2.1.1.1.1** contains node j_{13} with a $\mathsf{Rev}(k)$ action, we can use **S**$_{\neg,@}$ and **S**$_\bot$ to derive a contradiction. We first derive $\bot$ by applying **S**$_{\neg,@}$ to the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$, instantiating j with j_{13}. Then, we apply rule **S**$_\bot$. In general, we can always saturate constraint systems under universally quantified guarded trace formulas. This works because all trace formulas in a constraint system are guarded and the number of trace formulas derivable from a constraint system using just **S**$_{\neg,@}$ is finite.

System **1.1.2.2** is also contradictory, which we show Appendix A. Thus, we terminate without finding an attack and, as our search is complete, we therefore have a proof that $P \vDash_{E_{DH}} \varphi$.

The above example provides intuition for all rules except **N1**, **N5,6**, **N6**, and **N7**, which enforce normal-form message deduction. We explain them in the following paragraphs.

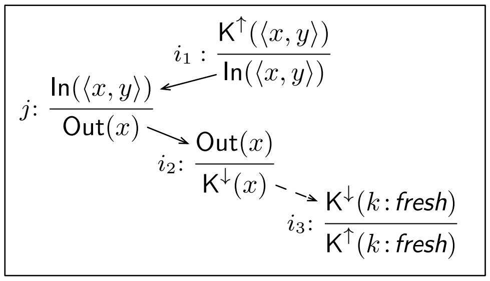

$i_1: \frac{\mathsf{K}^{\uparrow}(\langle x, y\rangle)}{\mathsf{In}(\langle x, y\rangle)}$
$j: \frac{\mathsf{In}(\langle x, y\rangle)}{\mathsf{Out}(x)}$
$i_2: \frac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}(x)}$
$i_3: \frac{\mathsf{K}^{\downarrow}(k\!:\!\mathit{fresh})}{\mathsf{K}^{\uparrow}(k\!:\!\mathit{fresh})}$

Figure 14. A contradictory constraint system where the adversary uses an instance of the protocol rule $\mathsf{In}(\langle x, y, \rangle)\text{-}[\,]\!\!\rightarrow\mathsf{Out}(x)$ to forward to himself the message x, which he deduced himself.

Rule **N1** ensures that all rule instances in node constraints are in $\downarrow_{DH}$-normal form, i.e., it allows us to prune constraint systems containing DH-reducible terms. Intuitively, this rule prevents inconsistent instantiations of variables occuring in the variants of a multiset rewriting rule. Consider for example the multiset rewriting rule $\mathsf{In}(x)\text{-}[\,]\!\!\rightarrow\mathsf{Out}(\mathrm{fst}(x))$. The corresponding DH,AC-variants are $\mathsf{In}(x)\text{-}[\,]\!\!\rightarrow\mathsf{Out}(\mathrm{fst}(x))$ and $\mathsf{In}(\langle y, z\rangle)\text{-}[\,]\!\!\rightarrow\mathsf{Out}(y)$. The rule **N1** allows pruning constraint systems where a node is labeled with the first variant and x is instantiated with a pair, as such constraint systems contradict the implicit assumption of the first rule variant, i.e., $\mathrm{fst}(x)$ is not DH-reducible. Pruning constraint systems with DH-reducible terms is crucial when reasoning about the variants of DH exponentiations. Rule **N7** provides further support for reasoning about DH exponentatiations, as it prunes constraint systems containing instances of deconstruction rules that can be replaced by instances of the construction rule for exponentiation.

Rule **N5,6** ensures that each $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-fact is derived, and therefore solved, at most once. Moreover, it ensures that every $\mathsf{K}^{\uparrow}$-premise deriving the same message as a $\mathsf{K}^{\downarrow}$-conclusion occurs after the $\mathsf{K}^{\downarrow}$-conclusion. This is required for Theorem 3 and allows pruning some constraint systems.

Rule **N6** prunes constraint systems where the adversary forwards a message via the protocol to himself. Such constraint systems occur when unfolding a deconstruction chain until it starts from a message variable whose content is received from the adversary. This is best seen in an example.

Example 8. Consider solving the premise $\mathsf{K}^{\uparrow}(k\!:\!\mathit{fresh})$ in the context of a protocol that contains the multiset rewriting rule $\mathsf{In}(\langle x, y, \rangle)\text{-}[\,]\!\!\rightarrow\mathsf{Out}(x)$. One of the cases that we must consider is captured by the constraint system depicted in Figure 14. It states that the adversary might deduce k using a deconstruction chain starting from the message x sent by node j. Note that we suppress the exponentiation tags, as they are irrelevant for this example. This constraint system is contradictory because, in all its solutions, the adversary must deduce $\mathsf{K}^{\uparrow}(x)$ before $\mathsf{K}^{\uparrow}(\langle x, y\rangle)$, which implies that he cannot deduce $\mathsf{K}^{\downarrow}(x)$ afterwards, as required by node i_3. We show that this constraint system is contradictory using rule **DG1**$_1$ after we used rule **N6** to derive $i_3 \lessdot i_1$.

	Protocol	Security Model	Result	Time [s]
1.	DH2 [31]	weakened eCK [31]	proof	6.7
2.	KAS1 [30]	KI+KCI [31]	proof	0.3
3.	KAS2 [30]	weakened eCK [31]	proof	2.9
4.	KAS2 [30]	eCK	attack	0.4
5.	KEA+ [5]	KI+KCI	proof	0.5
6.	KEA+ [5]	KI+KCI+wPFS	attack	0.6
7.	NAXOS [2]	eCK	proof	5.2
8.	NAXOS [2]	eCK+PFS	attack on PFS	4.8
9.	SIG-DH	PFS	proof	0.4
10.	SIG-DH	eCK	attack	0.6
11.	STS-MAC [3]	KI, reg-PK	UKS-attack	2.7
12.	STS-MAC-fix1 [3]	KI, reg-PK (with PoP)	proof	8.6
13.	STS-MAC-fix2 [3]	KI, reg-PK	proof	1.9
14.	TS1-2004 [22]	KI	UKS-attack	0.2
15.	TS1-2008 [29]	KI	proof	0.2
16.	TS2-2004 [22]	KI+wPFS	attack on wPFS	0.4
17.	TS2-2008 [29]	KI+wPFS	proof	0.7
18.	TS3-2004/08 [22], [29]	KI+wPFS	non-termination	-
19.	UM [28]	wPFS	proof	0.7
20.	UM [28]	PFS	attack	0.4

Table I
RESULTS OF CASE STUDIES

VII. Case Studies

We implemented our constraint-solving algorithm in a tool, called the TAMARIN prover. It provides both a command-line interface and a graphical user interface, which allows to interactively inspect and construct attacks and proofs. We evaluated our algorithm on numerous protocols. Table I lists the results, run on a laptop with an Intel i7 Quad-Core processor. The tool and all models are available at [19].

We modeled the Signed Diffie-Hellman (SIG-DH) protocol, the STS protocol and two fixes [3], the UM [28], KEA+ [5], and NAXOS [2] protocols, and the TS1, TS2, and TS3 protocols [22] and their updated versions [29]. We also modeled NIST's KAS1 and KAS2 protocols [30] and the related DH2 protocol by Chatterjee et. al. [31]. For each protocol, we formalized its intended and related security models and analyzed them using TAMARIN. For example, to verify Key Independence (KI) for STS, we model that the adversary can reveal certain session keys. Additionally, the adversary can register public keys for himself, even if those keys have been previously registered for another identity. In this example, we find the UKS attack reported in [3]. The first fix from [3] requires a Proof-of-Possession (PoP) of the private key for registering a public key. The second fix includes the identities of the participants in the signatures. We model and successfully verify both fixes. For NIST's KAS1 and KAS2 protocols [30], our analysis confirms both the security proof and the informal statements made in [31].

Our results indicate that, in general, TAMARIN is effective and efficient. For example, it requires 5.2 seconds to verify NAXOS in the eCK model (see Figure 3).

In general, there are two sources of non-termination of our algorithm. First, if the protocol contains loops (e.g., a rule like $\mathsf{A}(x)\text{-}[\,]\!\!\rightarrow\mathsf{A}(\mathsf{h}(x))$), then the reduction rule **DG2**$_{2,P}$ can be applied infinitely often during backwards search. Reasoning about such protocols requires support for loop invariants, as used in program verification. Second, if the protocol can serve as a generic message deduction oracle, then our normal-form conditions may fail to eliminate sufficiently many redundant steps. This explains the non-termination for the TS3-2004/08 protocols. It remains future work to develop normal-form conditions that improve reasoning about such protocols.

VIII. Related work

Corin et. al. [32] and Armando et. al. [33] use linear temporal logics and constraint solving for security protocol verification for a bounded number of sessions. Chevalier et. al. [8] and Shmatikov [9] prove that secrecy is decidable for a bounded number of sessions for DH theories similar to ours. Meadows et. al. [34] and Kapur et. al. [35] present unification algorithms for a DH theory similar to ours. In [35], Kapur et. al. show the undecidability of unification modulo a DH theory that also allows addition of exponents.

[13]–[15] support verification for an unbounded number of sessions, but do not consider inverses. Blanchet et. al. [13] extend ProVerif [36] to handle the property that $(x\hat{}y)\hat{}z \simeq (x\hat{}z)\hat{}y$. Goubault-Larrecq [14] accounts for this property using a Horn-theory approach and resolution modulo AC. Escobar et. al. [15] use Maude-NPAand equational unification to analyze secrecy properties of DH protocols. Since Maude-NPA supports user-specified equational theories, the verification problem with respect to our DH theory can be specified. It is however unclear if Maude-NPA can achieve unbounded verification for such a theory. In the free term algebra, Basin and Cremers [16] present models and tool support for compromising adversaries, based on Scyther [37].

Küsters and Truderung [7] give a transformation that, given a Horn theory modeling secrecy and simple authentication properties modulo a DH theory with inverses, produces a Horn theory in the free algebra, which they analyze using ProVerif. Their reduction is similar to our reduction from E_{DH} to AC, but works only for Horn clauses with ground exponents. As stated in [7], stronger security properties often violate this restriction. Since our approach allows for non-ground exponents, we can also find attacks where the adversary sends products, e.g., a protocol that receives a message x and leaks a secret if $\mathsf{g}\,\hat{}\,(\mathsf{a} * \mathsf{b} * \mathsf{x}^{-1}) = \mathsf{g}$.

Lynch and Meadows [10] and Mödersheim [11] give reductions for DH reasoning without inverses to reasoning modulo a simpler equational theory for a restricted class of protocols. Both require that all exponents used by a protocol remain secret forever. This excludes modeling ephemeral key reveals and thus verifying recent AKE protocols. Ngo et. al. [12] propose a method for the automated construction of computational proofs for a restricted class of DH-based protocols.

IX. Conclusion

We gave a novel constraint-solving algorithm and demonstrated its effectiveness in non-trivial case studies. Our algorithm exploits a special representation of message deduction that satisfies a “deconstruction chain” property, inspired by the theory underlying Athena [38] and Scyther [37], [39]. This property is key for achieving unbounded verification. It enables a backwards exploration of the interleavings of protocol steps guided by the solving of message deduction constraints. We constructed this representation by imposing normal-form conditions on the use of the (finite) variants of the message deduction rules. This construction and our algorithm should also work for other theories with the finite variant property, e.g., theories for XOR and blind-signatures.

Although we were motivated by the verification of AKE protocols, neither our protocol model nor our constraint-solving algorithm are tailored to them. We expect both our model and algorithm to be applicable to a wide range of security protocol verification problems. Due to our multiset-rewriting model, our approach is especially promising for verifying protocols that exploit internal state, which are often hard to analyze using the Horn-theory approach. We plan to investigate such stateful protocols in future work together with support for loop invariants.

References

[1] B. Schmidt, S. Meier, C. Cremers, and D. A. Basin, “Automated analysis of Diffie-Hellman protocols and advanced security properties,” in *Proceedings of the 25rd IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge MA, USA, June 25-27, 2012*. IEEE Computer Society, 2012, to be published.

[2] B. LaMacchia, K. Lauter, and A. Mityagin, “Stronger security of authenticated key exchange,” in *Provable Security*, ser. LNCS, vol. 4784. Springer, 2007, pp. 1–16.

[3] S. Blake-Wilson and A. Menezes, “Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol,” in *Proceedings of the 2nd International Workshop on Practice and Theory in Public Key Cryptography*. Springer, 1999, pp. 154–170.

[4] C. Cremers, “Session-StateReveal is stronger than eCK’s EphemeralKeyReveal: Using automatic analysis to attack the NAXOS protocol,” *International Journal of Applied Cryptography (IJACT)*, vol. 2, pp. 83–99, 2010.

[5] K. Lauter and A. Mityagin, “Security analysis of KEA authenticated key exchange protocol,” in *PKC 2006*, ser. LNCS, vol. 3958. Springer, 2006, pp. 378–394.

[6] M. Just and S. Vaudenay, “Authenticated multi-party key agreement,” in *ASIACRYPT 1996*, ser. LNCS, vol. 1163. Springer, 1996, pp. 36–49.

[7] R. Küsters and T. Truderung, “Using ProVerif to analyze protocols with Diffie-Hellman exponentiation,” in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2009, pp. 157–171.

[8] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents," in *FSTTCS 2003*, ser. LNCS, vol. 2914. Springer, 2003, pp. 124–135.

[9] V. Shmatikov, "Decidable analysis of cryptographic protocols with products and modular exponentiation," in *Programming Languages and Systems*, ser. LNCS. Springer, 2004, vol. 2986, pp. 355–369.

[10] C. Lynch and C. Meadows, "Sound approximations to Diffie-Hellman using rewrite rules," in *Information and Communications Security*, ser. LNCS. Springer, 2004, vol. 3269, pp. 65–70.

[11] S. Mödersheim, "Diffie-Hellman without difficulty," in *Proceedings of the 8th International Workshop on Formal Aspects of Security & Trust (FAST)*, 2011.

[12] L. Ngo, C. Boyd, and J. Nieto, "Automated proofs for Diffie-Hellman-based key exchanges," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, 2011, pp. 51 –65.

[13] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.

[14] J. Goubault-Larrecq, M. Roger, and K. Verma, "Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically," *Journal of Logic and Algebraic Programming*, vol. 64, no. 2, pp. 219–251, 2005.

[15] S. Escobar, C. Meadows, and J. Meseguer, "State space reduction in the Maude-NRL protocol analyzer," in *Computer Security - ESORICS 2008*, ser. LNCS. Springer, 2008, vol. 5283, pp. 548–562.

[16] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *Computer Security - ESORICS 2010*, ser. LNCS, vol. 6345. Springer, 2010, pp. 340–356.

[17] ——, "Degrees of security: Protocol guarantees in the face of compromising adversaries," in *19th EACSL Annual Conference on Computer Science Logic (CSL)*, ser. LNCS, vol. 6247. Springer-Verlag, 2010, pp. 1–18.

[18] H. Andréka, I. Németi, and J. van Benthem, "Modal languages and bounded fragments of predicate logic," *Journal of Philosophical Logic*, vol. 27, pp. 217–274, 1998.

[19] S. Meier and B. Schmidt, "The TAMARIN prover: source code and case studies," April 2012, available http://hackage.haskell.org/package/tamarin-prover-0.4.0.0.

[20] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol," in *Advances in Cryptology–CRYPTO 2005*, ser. LNCS, vol. 3621. Springer, 2005, pp. 546–566.

[21] C. Cremers, "Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 80–91.

[22] I. R. Jeong, J. Katz, and D. H. Lee, "One-round protocols for two-party authenticated key exchange," in *Applied Cryptography and Network Security*. Springer, 2004, pp. 220–232.

[23] F. J. Thayer, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct," *Journal of Computer Security*, vol. 7, no. 1, 1999.

[24] S. Escobar, R. Sasse, and J. Meseguer, "Folding variant narrowing and optimal variant termination," *Rewriting Logic and Its Applications*, pp. 52–68, 2010.

[25] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," *Term Rewriting and Applications*, pp. 294–307, 2005.

[26] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, vol. 28, pp. 119–134, 2003.

[27] E. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with Brutus," *ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, pp. 443–487, 2000.

[28] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman key agreement protocols," in *Selected Areas in Cryptography*, ser. LNCS. Springer, 1999, vol. 1556, pp. 630–630.

[29] I. R. Jeong, J. Katz, and D. H. Lee, "One-round protocols for two-party authenticated key exchange (full)," 2008, http://www.cs.umd.edu/~jkatz/papers/1round_AKE.pdf.

[30] *SP 800-56B, Special Publication 800-56B, Recommendation for Pair-Wise Key Establishment Schemes using Integer Factorization Cryptography*, National Institute of Standards and Technology, August 2009.

[31] S. Chatterjee, A. Menezes, and B. Ustaoglu, "A generic variant of NIST's KAS2 key agreement protocol," in *Proceedings of the 16th Australasian conference on Information security and privacy (ACISP'11)*. Springer, 2011, pp. 353–370.

[32] R. Corin, A. Saptawijaya, and S. Etalle, "A logic for constraint-based security protocol analysis," in *Proceedings of IEEE Symposium on Security and Privacy, S&P 2006*, 2006, pp. 155–168.

[33] A. Armando, R. Carbone, and L. Compagna, "LTL model checking for security protocols," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, 2007, pp. 385 – 396.

[34] C. Meadows and P. Narendran, "A unification algorithm for the Group Diffie-Hellman protocol," in *Proc. of WITS 2002*, 2002.

[35] D. Kapur, P. Narendran, and L. Wang, "An E-unification algorithm for analyzing protocols that use modular exponentiation," in *Rewriting Techniques and Applications*. Springer, 2003, pp. 165–179.

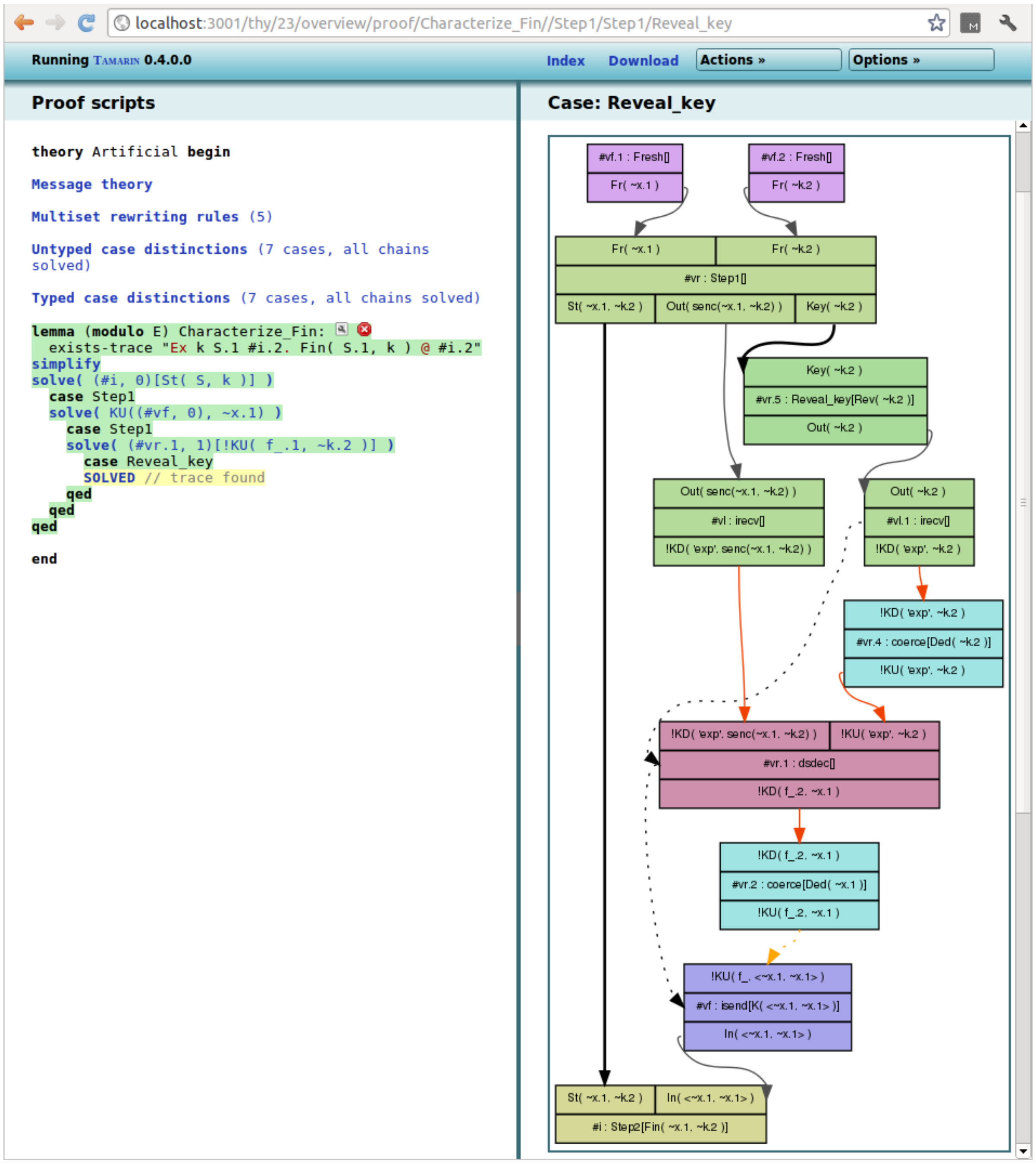
[36] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE, 2001, pp. 82–96.

[37] C. Cremers, “The Scyther Tool: Verification, falsification, and analysis of security protocols,” in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 5123. Springer, 2008, pp. 414–418.

[38] D. Song, S. Berezin, and A. Perrig, “Athena: A novel approach to efficient automatic security protocol analysis,” *Journal of Computer Security*, vol. 9, pp. 47–74, 2001.

[39] S. Meier, C. Cremers, and D. A. Basin, “Strong invariants for the efficient construction of machine-checked protocol security proofs,” in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2010, pp. 231–245.

[40] S. Doghmi, J. Guttman, and F. Thayer, “Searching for shapes in cryptographic protocols,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, O. Grumberg and M. Huth, Eds. Springer, 2007, vol. 4424, pp. 523–537.



```
theory Artificial begin

Message theory

Multiset rewriting rules (5)

Untyped case distinctions (7 cases, all chains
solved)

Typed case distinctions (7 cases, all chains solved)

lemma (modulo E) Characterize_Fin:
  exists-trace "Ex k S.1 #i.2. Fin( S.1, k ) @ #i.2"
simplify
solve( (#i, 0)[St( S, k )] )
  case Step1
  solve( KU((#vf, 0), ~x.1) )
    case Step1
    solve( (#vr.1, 1)[!KU( f_.1, ~k.2 )] )
      case Reveal_key
      SOLVED // trace found
    qed
  qed
qed

end
```

Figure 15. Screenshot of the constraint-solving visualization provided by the TAMARIN prover. Here we use it to characterize $\exists x\ k\ i.\ \mathsf{Fin}(x,k)@i$ for our example protocol P from Example 1. We explain the idea behind such a characterization at the end of Appendix A. The left half provides an overview of the steps taken by our constraint-solving algorithm. The steps stem from trying to prove that there exists a trace such satisfying $(\exists x\ k\ i.\ \mathsf{Fin}(x,k)@i)$. The desired characterization consists of all leaves marked with "SOLVED". The corresponding constraint systems cover all possible P-solutions of this property. In the right half, a visualization of the constraint system corresponding to the step marked with a yellow background is shown. It corresponds to System **1.1.2.1.1.1.1** from Figure 16. The orange, dashed arrow from node `#vr.1` to `#vf` denotes an implicit construction dependency. The black, dotted arrows denote additional ordering constraints on the nodes. They are implied by the property **N6**. The solid arrows denote edge constraints. Their width and color depends on the type of fact that they connect.

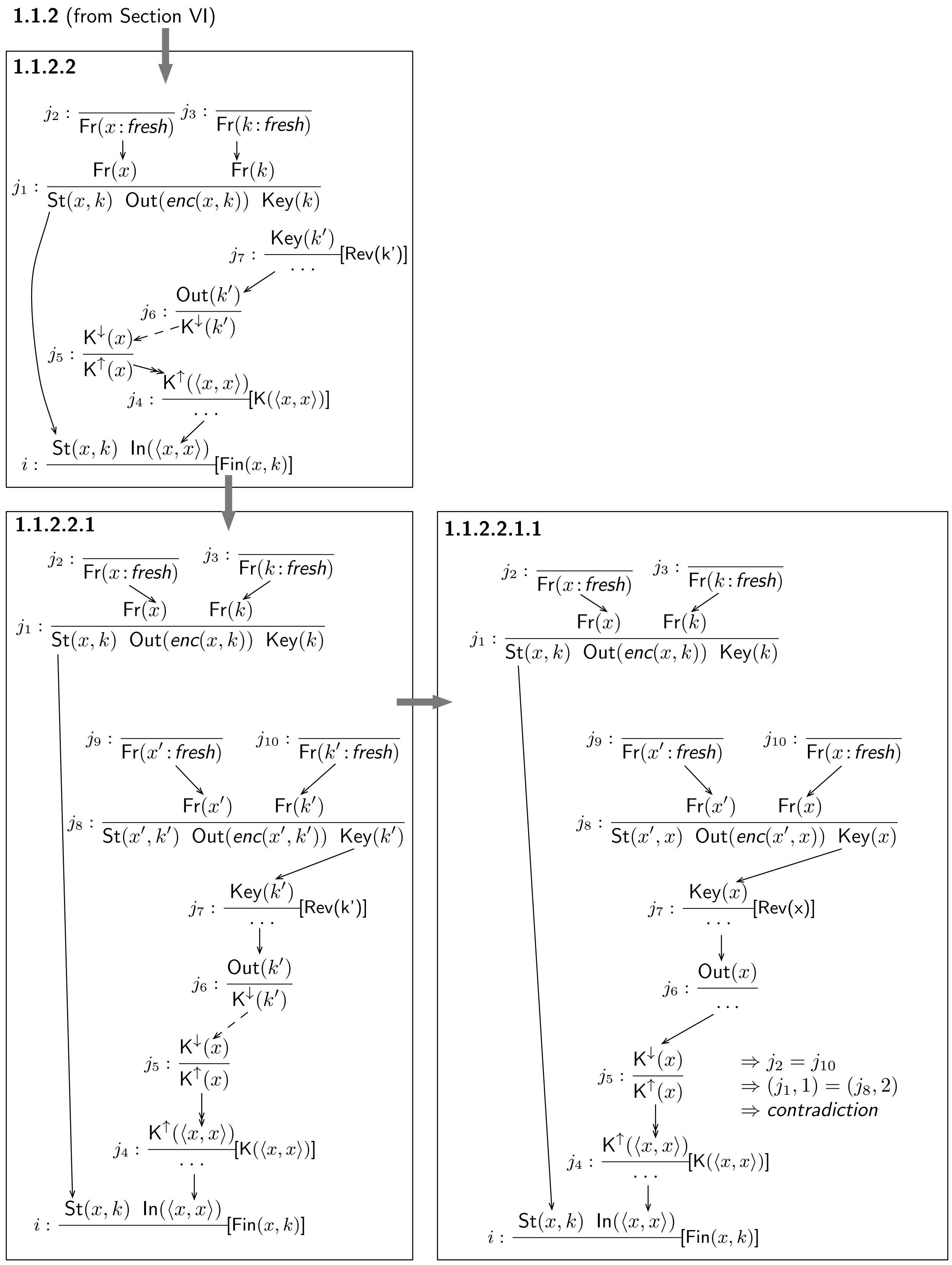

1.1.2 (from Section VI)
1.1.2.2
1.1.2.2.1
1.1.2.2.1.1
j_1 : Fr(x) Fr(k) / St(x,k) Out(enc(x,k)) Key(k)
j_2 : Fr(x : fresh)
j_3 : Fr(k : fresh)
j_4 : K$^\uparrow$($\langle x,x\rangle$) [K($\langle x,x\rangle$)]
j_5 : K$^\downarrow$(x) / K$^\uparrow$(x)
j_6 : Out(k') / K$^\downarrow$(k')
j_7 : Key(k') [Rev(k')]
j_8 : Fr(x') Fr(k') / St(x',k') Out(enc(x',k')) Key(k')
j_9 : Fr(x' : fresh)
j_{10} : Fr(k' : fresh)
j_{10} : Fr(x : fresh)
j_8 : Fr(x') Fr(x) / St(x',x) Out(enc(x',x)) Key(x)
j_7 : Key(x) [Rev(x)]
j_6 : Out(x)
i : St(x,k) In($\langle x,x\rangle$) [Fin(x,k)]
$\Rightarrow j_2 = j_{10}$
$\Rightarrow (j_1, 1) = (j_8, 2)$
$\Rightarrow$ contradiction

Figure 16. Remaining constraint-reduction steps from Example 7. We explain these remaining steps in Appendix A. The "..." at either end of an edge refers to the fact at the other end.

Appendix A.
Remainder of Constraint-Solving Example

In this section, we finish Example 7 from Section VI. In System **1.1.2** from Figure 11, the $\mathsf{Out}(y)$-premise of node j_6 is open. Solving it results in two cases. We already explained the first one in Example 7 and explain the second one here.

The constraint system for the second case is depicted in Figure 16 as System **1.1.2.2**. It states that the required Out-premise might stem from the conclusion of the instance of the rule of protocol P, defined in Example 1, that allows a key reveal. After solving the open $\mathsf{Key}(k')$ premise and reducing the deconstruction chain, we obtain the trivially contradictory System **1.1.2.2.1**. Note that $j_2 = j_{10}$ is contradictory, as it requires the edge from the $\mathsf{Fr}(x)$-fact to end both in a first and a second premise of a multiset rewriting rule.

Note that that we can also use a variant of our algorithm to characterize all possible executions of a protocol that exhibit a certain structure; thereby generalizing the work on characterizing shapes in strand spaces [40] to labeled multiset rewriting systems and guarded trace properties. The variant of our algorithm works such that it runs as long as unsolved constraint systems remain in its state. The remaining solved constraint systems characterize all executions that satisfy the initial constraint system. For our example, characterizing the executions of P that satisfy $\exists x\ k\ i.\ \mathsf{Fin}(x,k)@i$ results in the single solved constraint system equal to System **1.1.2.1.1.1.1** without the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j)$. Figure 15 shows a screenshot of the characterization of $\exists x\ k\ i.\ \mathsf{Fin}(x,k)@i$ for our example protocol P as computed by our implementation.

Appendix B.
Explanation of Constraint-Reduction Rules

Recall that our constraint-reduction rules are designed such that we can extract a P-solution from every solved constraint system. To understand the rules, it therefore helps to focus on the partial solution of a constraint system Γ. We define the *partial solution of* Γ as the set of all nodes, edges, timepoint orderings, and implicit construction dependencies of Γ. Informally, the partial solution of a solved constraint system Γ is converted to a P-solution of Γ by first instantiating all variables with distinct names and trace indices, chosen according to their sort, and then expanding the implicit construction constraints. The fact that no rule is applicable to a solved constraint system guarantees that this conversion succeeds and results in a P-solution.

Our rule naming scheme embodies this design: the name of a rule refers to the form of guarded trace formulas that it solves or the property of normal dependency graphs that it ensures. If there are several rules responsible for ensuring a property, then they are distinguished by subscripts.

We now explain each of the three groups of rules from Figure 10. Afterwards, we explain the strategy that our algorithm uses to decide when to apply which rule.

A. Trace Formula Reduction Rules

The first group of rules ensures that the partial solution satisfies the guarded trace formulas in the constraint system. All of these rules have side-conditions that ensure that they cannot be applied multiple times with the same parameters. The interaction between guarded trace formulas and graph constraints happens via the rules $\mathtt{S}_{\approx}$, $\mathtt{S}_{\doteq}$, $\mathtt{S}_{@}$, $\mathtt{S}_{\neg,@}$, $\mathtt{S}_{\neg,\lessdot}$, and $\mathtt{S}_{\forall}$.

The rule $\mathtt{S}_{\approx}$ solves an equality between two terms by performing a case distinction over all possible AC-unifiers. The rule $\mathtt{S}_{\doteq}$ solves an equality between two temporal variables by substituting all occurrences of one by the other. The rule $\mathtt{S}_{@}$ solves an action constraint $f@i$ by introducing a new node constraint $i : ru$ for every protocol rule ru that could give rise to the action f. The rules $\mathtt{S}_{\neg,\approx}$ and $\mathtt{S}_{\neg,\doteq}$ derive contradictions from negated equalities. The rule $\mathtt{S}_{\neg,@}$ derives a contradiction from a negated action. The rule $\mathtt{S}_{\neg,\lessdot}$ performs a case distinction to solve negated timepoint ordering constraints that do not trivially hold.

The rule $\mathtt{S}_{\bot}$ removes contradictory constraint systems. The rule $\mathtt{S}_{\vee}$ performs a case distinction to solve a disjunction. The rule $\mathtt{S}_{\wedge}$ adds the conjuncts of a conjunction to the constraint system. The rule $\mathtt{S}_{\exists}$ ensures that, for every existential quantification $\exists x{:}s.\varphi$, there is at least one term w of sort s such that $\varphi\{w/x\}$ holds. Intuitively, the rule $\mathtt{S}_{\forall}$ saturates the constraint system under the guarded trace formulas and the actions of nodes in the partial solution. It introduces the guarded trace formula $\psi\sigma$ for every guarded trace formula $\forall\vec{x}.\neg(f@i) \vee \psi$ whose guard $f@i$ matches an action of the constraint system under the substitution σ. It checks whether $\psi\sigma \notin_{AC} \Gamma$ modulo renaming of bound variables to avoid introducing the same guarded trace formula multiple times. This check suffices, as we never remove a guarded trace formula from the constraint system.

B. Graph Constraint-Reduction Rules

The second group of rules ensures that the partial solution satisfies all properties of dependency graphs. The rule $\mathtt{U}_{lbl}$ ensures nodes are uniquely labeled. The rule $\mathtt{DG1}_1$ prunes constraint systems where $\lessdot_\Gamma$ is not a strict partial order. The rule $\mathtt{DG1}_2$ ensures that edges connect facts that are equal modulo AC. The rule $\mathtt{DG2}_1$ ensures that each premise has at most one incoming edge. The rule $\mathtt{DG2}_{2,P}$ ensures that each premise has at least one incoming edge. The definition of open premises ensures that this rule is not applied twice to the same premise. We exclude $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-facts from being solved with this rule, as they must be solved with the special rules for solving message deduction constraints to avoid non-termination. The rule $\mathtt{DG3}$ ensures that each linear conclusion has at most one outgoing edge. The rule $\mathtt{DG4}$ ensures that instances of the FRESH rule are unique.

Note that the rules $\mathtt{U}_{lbl}$ and $\mathtt{DG1}_2$ rely on the rule $\mathtt{S}_{\approx}$ for solving the introduced equalities. Provided that these equalities are solved eagerly, repeatedly applying only rules from this group, except $\mathtt{DG2}_{2,P}$, is guaranteed to terminate. Repeatedly applying rule $\mathtt{DG2}_{2,P}$ is not guaranteed to terminate for all protocols. It can for example be applied infinitely often, when reasoning about the protocol

$$\{[]\text{-}[]\!\!\rightarrow \mathsf{A}(1),\ \mathsf{A}(x)\text{-}[]\!\!\rightarrow \mathsf{A}(x)\}\ .$$

Our constraint-reduction relation must be extended with additional rules to ensure termination for such protocols. Nevertheless, for typical security protocols, the rule $\mathtt{DG2}_{2,P}$ suffices to reason about their execution, as the dependency relation between facts other than $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-facts is well-founded.

C. Message Deduction Constraint-Reduction Rules

The third group of rules allows reasoning about message deduction constraints. The rules $\mathtt{DG2}_{2,\uparrow i}$ and $\mathtt{DG2}_{2,\uparrow e}$ solve open $\mathsf{K}^{\uparrow}$-premises. The rules $\mathtt{DG2}_{2,\downarrow}$ and $\mathtt{DG2}_{\dashrightarrow}$ solve open $\mathsf{K}^{\downarrow}$-premises. All four rules can be seen as refined versions of the $\mathtt{DG2}_{2,P}$ rule, which exploit our normal-form conditions to improve efficiency and avoid non-termination. The rules $\mathtt{N1}$, $\mathtt{N5,6}$, and $\mathtt{N6}$ ensure that the partial solution satisfies the normal-form conditions **N1,5,6**. There are no rules for ensuring the normal-form conditions **N2-4** because, as we will see, they hold by construction. All of these rules work together to solve message deduction constraints. Their interplay can be observed well in Example 7. Here, we explain each rule individually.

The rule $\mathtt{DG2}_{2,\uparrow e}$ solves an open $\mathsf{K}^{\uparrow}_{e}(m)$-premise p by enumerating all rules that have a unifying $\mathsf{K}^{\uparrow}$-conclusion. The side condition $\{m\} = inp(m)$ ensures that m is not a pair, inverse, or product, as such $\mathsf{K}^{\uparrow}$-premises are handled by the rule $\mathtt{DG2}_{2,\uparrow i}$. The rule is only applied to non-trivial messages, as we can always introduce the incoming edges of trivial $\mathsf{K}^{\uparrow}$-premises when extracting a solution from a solved constraint system. The requirement that the open premise is equal to the added conclusion is ensured indirectly via the added edge constraint $(i, 1) \rightarrowtail p$ and the rule $\mathtt{DG1}_2$.

The rule $\mathtt{DG2}_{2,\uparrow i}$ solves an open implicit m-construction p by enumerating all rewriting rules that have a unifying $\mathsf{K}^{\uparrow}$-conclusion. This rule exploits Lemma 3 to directly enumerate the possible deductions of input components of messages known to the adversary. Compared to the rule $\mathtt{DG2}_{2,\uparrow e}$, this rule avoids an exponential blowup when solving pair $\mathsf{K}^{\uparrow}$-premises, as it avoids (for each nesting of the pair constructor) the case distinction whether the unifying $\mathsf{K}^{\uparrow}$-conclusion stems from the PAIR ↑ or the COERCE message deduction rule. Similarly, it leads to efficiency improvements when solving inverse and product $\mathsf{K}^{\uparrow}$-premises.

Rule $\mathtt{DG2}_{2,\downarrow}$ solves an open $\mathsf{K}^{\downarrow}$-premise by introducing an arbitrary instance of the IRECV rule from which the adversary deduced the $\mathsf{K}^{\downarrow}$-premise using a chain of deconstruction rules. Such an instance of the IRECV rule must exist due to Lemma 2. The possible messages sent by the protocol from which the chain could start are enumerated using the rule $\mathtt{DG2}_{2,P}$ on the newly introduced $\mathsf{Out}(t)$ premise.

$$\begin{tikzcd}
 & \frac{\mathsf{Out}(\langle \mathsf{a},\mathsf{b}\rangle)}{\mathsf{K}^{\downarrow}(\langle \mathsf{a},\mathsf{b}\rangle)} \arrow[dl] \arrow[dr] & \\
\frac{\mathsf{K}^{\downarrow}(\langle \mathsf{a},\mathsf{b}\rangle)}{\mathsf{K}^{\downarrow}(\mathsf{a})} \arrow[d] & & \frac{\mathsf{K}^{\downarrow}(\langle \mathsf{a},\mathsf{b}\rangle)}{\mathsf{K}^{\downarrow}(\mathsf{b})} \arrow[d] \\
\frac{\mathsf{K}^{\downarrow}(\mathsf{a})}{\mathsf{K}^{\uparrow}(\mathsf{a})} \arrow[dr] & & \frac{\mathsf{K}^{\downarrow}(\mathsf{b})}{\mathsf{K}^{\uparrow}(\mathsf{b})} \arrow[dl] \\
 & \frac{\mathsf{K}^{\uparrow}(\mathsf{a}) \quad \mathsf{K}^{\uparrow}(\mathsf{b})}{\mathsf{K}^{\uparrow}(\langle \mathsf{a},\mathsf{b}\rangle)} &
\end{tikzcd}$$

Figure 17. An example of the deduction of $\mathsf{K}^{\uparrow}(\langle \mathsf{a},\mathsf{b}\rangle)$ from $\mathsf{Out}(\langle \mathsf{a},\mathsf{b}\rangle)$ in a normal dependency graph. Because of condition **N2**, the pair must be deconstructed first, which results in $\mathsf{K}^{\downarrow}$-conclusions deriving the same message as $\mathsf{K}^{\uparrow}$-conclusions.

Rule $\mathtt{DG2}_{\dashrightarrow}$ refines a deconstruction chain by unfolding its definition by one step. It makes a case distinction on whether a deconstruction chain consists of a single edge or an edge to a deconstruction rule ri and a further deconstruction chain starting from the conclusion of ri. We disallow refining chains that start from a message variable, as this would lead to non-termination.

Note that later constraint-reduction steps will often instantiate this message variable and allow us to finish refining the chain. This is always the case for protocols that check the types of encrypted or signed data. We can model such type checks using sorts. This is sound for any implementation that ensures that all bitstrings representing terms of one sort are different from all bitstrings representing terms of any other incomparable sort. To reason about protocols that receive arbitrary messages as cleartext, we use the rule $\mathtt{N6}$, as explained in Example 8. The constraint-reduction rules given here are not sufficient to reason about protocols that blindly forward encrypted messages; e.g., a protocol containing a rule like $\mathsf{In}(\mathrm{enc}(x, k_1)\text{-}[]\!\!\rightarrow\mathsf{Out}(\mathrm{enc}(x, k_2))$ for $x{:}msg$. For such protocols, we must prove an additional protocol specific invariant that characterizes the possible instantiations of the forwarded messages (see [39, Section 3-B] for such an invariant). This is outside the scope of this paper.

Rule $\mathtt{N1}$ ensures that all rule instances in node constraints are in $\downarrow_{DH}$-normal form, which holds due to property **N1**. Rule $\mathtt{N5,6}$ exploits that the K-conclusions of all rule instances, except the instances of the COERCE rule and the pair and the inversion construction rules, are unique, regardless of the deconstruction and exponentiation tags. The conclusions of instances of the COERCE rule and the pair and the inversion construction rules are not guaranteed to be unique because we enforce long-normal-form message deductions. See Figure 17 for an example. Nevertheless, even with long-normal-form deductions, every $\mathsf{K}^{\uparrow}$-premise deriving the same message as a $\mathsf{K}^{\downarrow}$-conclusion can only occur *after* this $\mathsf{K}^{\downarrow}$-conclusion. Rule **N6** exploits this property. This rule is important, as it allows us to prune constraint systems where the adversary forwards a message via the protocol to himself. Such constraint systems occur when unfolding a deconstruction chain until it starts from a message variable whose content is received from the adversary.

D. Rule Application Strategy

We currently use a very simple rule application strategy. Whenever possible, we apply a rule that is guaranteed not to result in a case distinction; i.e., we apply one of the rules $\mathbf{S}_{\perp}$, $\mathbf{S}_{\approx}$, $\mathbf{S}_{\doteq}$, $\mathbf{S}_{\neg,\approx}$, $\mathbf{S}_{\neg,\doteq}$, $\mathbf{S}_{\neg,@}$ $\mathbf{S}_{\wedge}$, $\mathbf{S}_{\exists}$, $\mathbf{S}_{\forall}$, $\mathbf{U}_{lbl}$, $\mathbf{DG1}_1$, $\mathbf{DG1}_2$, $\mathbf{DG2}_1$, **DG3**, **DG4**, **N1**, **N5,6**, or **N6**. Otherwise, we use the first applicable rule from the following list: $\mathbf{S}_{@}$, $\mathbf{S}_{\vee}$, $\mathbf{S}_{\neg,\lessdot}$, $\mathbf{DG2}_{2,P}$, $\mathbf{DG2}_{\dashrightarrow}$. $\mathbf{DG2}_{2,\downarrow}$, $\mathbf{DG2}_{2,\uparrow i}$, and $\mathbf{DG2}_{2,\uparrow e}$. These rules may result in a case distinction.

Often there are several open $\mathsf{K}^{\uparrow}$-premises in a constraint system. The order of solving them has a significant impact on the size of the generated proofs and therefore the runtime of the algorithm. We use the following heuristic to delay the solving of $\mathsf{K}^{\uparrow}$-premises that are unlikely to result in a contradiction. We delay solving a premise p requiring a message m, if m does not contain a variable of sort *fresh* or if m can be extracted using projection and inversion from a message sent by a protocol step that does not occur later than p. Intuitively, we delay solving such premises because the adversary is likely to be able to construct their required messages.

Appendix C. Proofs

As stated in Section IV, our approach supports the combination of DH, pairing, and an arbitrary subterm-convergent rewrite theory. We therefore perform all proofs with respect to an arbitrary equational theory with these properties and extend the definitions from the main body of the paper where necessary.

In the following, we assume given a signature Σ_{ST} where all function symbols have sort $msg \times \ldots \times msg \rightarrow msg$ and a subterm convergent rewriting system R_{ST} where all left-hand-sides are built from Σ_{ST} and $\mathcal{V}_{msg}$. Then

$$\Sigma_{DH} = \{\langle\_,\_\rangle, \mathrm{fst}(\_), \mathrm{snd}(\_), \_\,\hat{}\,\_, \_^{-1}, \_ * \_, 1\} \uplus \Sigma_{ST}$$

and E_{DH} is generated by R_{ST} and Equations (2–10) from Figure 2. Note that this covers our original definition for $\Sigma_{ST} = \{\mathrm{enc}(\_,\_), \mathrm{dec}(\_,\_), \mathrm{h}(\_)\}$ and $R_{ST} = \{\mathrm{dec}(\mathrm{enc}(m,k),k) \rightarrow m\}$.

Where convenient, we use mult, exp, and inv to denote the function symbols $*$, $\hat{}$, and $^{-1}$.

A. Background for Proofs

A *position* is a sequence of natural numbers. For a term t and a position p, we denote the *subterm of t at position p* with $t|_p$. Formally, $t|_p$ is defined as t if $p = [\,]$, $t_i|_{p'}$ if $p' = [i] \cdot p$ and $t = f(t_1, \ldots, t_k)$ for $1 \le i \le k$, and undefined otherwise. We say p is a *valid position in t* if $t|_p$ is defined. For two positions p and p', p *is above* p' if p is a proper prefix of p'. In this case, p' *is below* p. If p is neither above nor below nor equal to p', then both positions are *incomparable*. We say p and p' are *siblings* if $|p| = |p'|$, $p_i = p'_i$ for $1 \le i < |p|$, and $p_{|p|} \ne p'_{|p|}$.

B. Proofs for Section V-A

The following lemma formalizes the relation between the multiset-rewriting-based semantics given in Section IV and the dependency-graph-based semantics for the same protocol and message deduction rules.

Lemma 4 (Justification of the statement on page 6.)**.** *For all protocols P,*

$$traces(P) =_{E_{DH}} \{trace(dg) \mid dg \in dgraphs_{E_{DH}}(P \cup MD)\}\ .$$

Proof: We first define $cfacts(dg)$ as the multiset of consumable facts in a dependency graph, i.e., the facts of persistent conclusions and linear conclusion with no outgoing edges in dg. We prove both directions separately.

$\subseteq_{E_{DH}}$: First, we prove by induction over the multiset rewriting derivation $\emptyset^{\sharp} \xrightarrow{A_1}_P \ldots \xrightarrow{A_k}_P S_k$, that if each instance of the FRESH rule is used at most once, then there is $dg = (I, D) \in dgraphs_{E_{DH}}(P \cup MD)$ such that

$$cfacts(dg) =_{E_{DH}} S_k, \quad (1)$$
$$I_{j+1} = \text{-}[\,]\!\!\rightarrow \mathsf{Fr}(m) \text{ iff } S_{j+1} \setminus^{\sharp} S_j = \{\mathsf{Fr}(m)\}^{\sharp}, \quad (2)$$
$$\text{and } trace(dg) =_{E_{DH}} [A_1, \ldots, A_k]. \quad (3)$$

For the multiset rewriting derivation $\emptyset^{\sharp}$, the empty dependency graph $([\,], \emptyset)$ satisfies (1)–(3). Let $\emptyset^{\sharp} \xrightarrow{A_1}_P \ldots \xrightarrow{A_k}_P S_k$ and $dg = (I, D) \in dgraphs_{E_{DH}}(P \cup MD)$ such that (1)–(3) hold. Let A_{k+1} and S_{k+1} arbitrary such that $S_k \xrightarrow{A_{k+1}}_P S_{k+1}$ and the uniqueness condition for FRESH is not violated. Then there must be some rule $l' \text{-}[\ a'\ ]\!\!\rightarrow r' \in ginsts(P \cup MD \cup \{\text{FRESH}\})$ and $l \text{-}[\ a\ ]\!\!\rightarrow r =_{E_{DH}} l' \text{-}[\ a'\ ]\!\!\rightarrow r'$ such that (a) $lfacts(l) \subseteq^{\sharp} S_k$, (b) $pfacts(l) \subseteq set(S_k)$, (c) $A_{k+1} = set(a)$, and (d) $S_{k+1} = ((S_k \setminus^{\sharp} lfacts(l)) \cup^{\sharp} mset(r))$. We can extend dg with a node $k+1$ with rule instance $l' \text{-}[\ a'\ ]\!\!\rightarrow r'$ since (a), (b), and (1) ensure that we can add the required edges. We call this extension dg'. Note that we require equality modulo E_{DH} here for **DG1**. If the rule is an instance of FRESH, then (2) ensures that we do not violate **DG4**. Then (2) holds because the same FRESH instance has been added to both. Condition (3) holds because

$set(a) =_{E_{DH}} set(a')$, and (1) holds because $r =_{E_{DH}} r'$ and $lfacts(l) =_{E_{DH}} lfacts(l')$.

$\supseteq_{E_{DH}}$: For the other direction, we prove by induction over I that for all $dg = (I, D) \in dgraphs_{E_{DH}}(P \cup MD)$, there is $\emptyset^\sharp \xrightarrow{A_1}_P \ldots \xrightarrow{A_k}_P S_k$ such that (1)–(3) hold. The proof is analogous to the other direction. ∎

C. Proofs for Section V-B

To account for the modified definition of E_{DH}, we change the definition of DH to: DH is the union of R_{ST} and the rewriting system obtained by orienting equations (2–3,9–10) from Figure 2 and all equations from Figure 6 from left to right. Then DH is AC-coherent and AC-convergent, an equational presentation of E_{DH}, and E_{DH} has the finite variant property for this presentation. In the following, we denote the equations (1–10) from Figure 2 by (D1–D10) and the equations (1–10) from Figure 6 by (L1–L10).

The following Lemma formalizes the relation between dependency graphs modulo E_{DH} and dependency graphs modulo AC.

Lemma 5 (Justification of the statement on page 7)**.** *For all protocols P,*

$$\begin{aligned} & dgraphs_{E_{DH}}(P \cup MD)\big\downarrow_{DH} \\ =_{AC} & \{dg \mid dg \in dgraphs_{AC}(\lceil P \cup MD \rceil^{DH}) \\ & \wedge\ dg \downarrow_{DH}\text{-}normal\ \}. \end{aligned}$$

Proof: We prove both directions separately.

$\subseteq_{AC}$: We prove this by induction on the length k of the sequence of rule instances. If $k = 0$, then the empty dependency graph is also an element of the set on the right-hand-side. Let $dg \in dgraphs_{E_{DH}}(P \cup MD)$ such that $dg\downarrow_{DH} \in_{AC} dgraphs_{AC}(\lceil P \cup MD \rceil^{DH})$. Let dg' an extension of dg with $ri \in ginsts(P \cup MD \cup \{\text{FRESH}\})$ and the required edges such that **DG4** is not violated. Then $dg'\downarrow_{DH}$ contains the new rule instance $ri\downarrow_{DH}$ and is $\downarrow_{DH}$-normal. To see that $dg'\downarrow_{DH} \in_{AC} dgraphs_{AC}(\lceil P \cup MD \rceil^{DH})$, note that equality of edge source and target modulo E_{DH} implies equality modulo AC after normalization. To see that $ri\downarrow_{DH} \in_{AC} ginsts(\lceil P \cup MD \rceil^{DH} \cup \{\text{FRESH}\})$, we distinguish two cases. First, if ri is an instance of FRESH, then ri is already $\downarrow_{DH}$-normal, $ri\downarrow_{DH} \in_{AC} ginsts(\text{FRESH})$, and **DG4** is not violated. Second, if ri is not an instance of FRESH, then there is $ru \in P \cup MD$ with $ru\sigma = ri$. Hence, there is a variant $ru' \in \lceil P \cup MD \rceil^{DH}$ and a substitution σ' such that $ri\downarrow_{DH} = (ru\sigma)\downarrow_{DH} =_{AC} ru'\sigma'$ and hence $ri\downarrow_{DH} \in_{AC} ginsts(\lceil P \cup MD \rceil^{DH})$.

$\supseteq_{AC}$: We prove this by induction on the length k of the sequence of rule instances. If $k = 0$, then the empty dependency graph is clearly also an element of the set on the left-hand-side. Let $dg \in dgraphs_{AC}(\lceil P \cup MD \rceil^{DH})$ and $dg \downarrow_{DH}$-normal such that there is $\hat{dg} \in dgraphs_{E_{DH}}(P \cup MD)$ with $\hat{dg}\downarrow_{DH} =_{AC} dg$. Let dg' be an extension of dg with $ri \in ginsts(\lceil P \cup MD \rceil^{DH} \cup \{\text{FRESH}\})$ and the required edges such that $ri \downarrow_{DH}$-normal and **DG4** not violated. The FRESH case is straightforward. In the other case, there is $ru \in P \cup MD$ and σ such that $ri =_{E_{DH}} ru\sigma$. Hence, we can extend $\hat{dg}$ with $ru\sigma \in ginsts(P \cup MD)$ and obtain $\hat{dg}' \in dgraphs_{E_{DH}}(P \cup MD)$ with $\hat{dg}'\downarrow_{DH} =_{AC} dg'$. ∎

D. Proofs for Section V-C

1) $*$-restricted Protocols: To account for the additional reducible function symbols in Σ_{ST}, we change condition (a) in the definition of $*$-restricted protocols to: l does not contain $*$, $\hat{}$, ${}^{-1}$, fst, snd, and function symbols f such that there is $s \to t \in R_{ST}$ with $root(s) = f$.

The definition of $*$-restricted protocols is motivated by the idea that we can restrict ourselves to protocols that do not "generate new products". Because we want to allow unrestricted usage of exponentiation and inversion, this is impossible to achieve. Instead, the definition ensures that newly generated products only combine factors that have been generated earlier. For products that occur in exponent-position, we cannot guarantee this, but these cannot be extracted anyways. To formally prove this result, we first define the right notion of *factors*.

Definition 1. The *factors* of a term t are defined as follows.

$$factors(t) = \begin{cases} factors(u) \cup factors(v) & \text{if } t = u * v \\ factors(s) & \text{if } t = s^{-1} \\ \{t, t^{-1}\} & \text{otherwise} \end{cases}$$

Since we cannot forbid all occurrences of products, we first define a set of positions where no term can be extracted from.

Definition 2. A position p is an *exponent position* if $root(t|_{p'}) = \exp$ and $p' \cdot [2]$ is either above or equal to p. We extend this definition to facts in the expected way, i.e., for a fact $F(t_1, \ldots, t_k)$, the position $[i] \cdot p$ is an exponent position if p is an exponent position in t_i. Analogously, we extend the definition to sequences of facts and multiset rewriting rules. We say a position is *accessible* if it is not an exponent position.

Definition 3. An *accessible product position* in a term t is an accessible position p in t such that $t|_p$ is a product. We use $appos(t)$ to denote this set of positions. The *accessible factors* of a term t are then defined as $afactors(t) = \bigcup_{p \in appos(t)} factors(t|_p)$. Analogously, we define the *accessible variable positions* p in t such that $t|_p \in \mathcal{V}_{msg}$ and denote them with $avpos(t)$. The *accessible variables* of a term t are then defined as $avars(t) = \bigcup_{p \in avpos(t)} \{t|_p\}$. We extend these notions to facts, sequences of facts, and multiset rewrite rules in the expected way.

We can now define the required condition on multiset rewriting rules that do not "construct accessible products". Note that since we want to allow unrestricted usage of exponentiation and inversion, the condition is slightly weaker and only ensures that such rules do not create accessible products with new factors. This is exactly the property we require to simplify the message deduction.

Definition 4. A multiset rewriting rule $l\text{-}[\ a\ ]\!\!\rightarrow r$ is *factor-restricted* if for all $\downarrow_{DH}$-normal substitutions σ,

$$\mathit{afactors}((r\sigma)\downarrow_{DH}) \subseteq_{AC} \mathit{afactors}((l\sigma)\downarrow_{DH}).$$

A protocol P is factor-restricted if all $l\text{-}[\ a\ ]\!\!\rightarrow r \in P$ are factor-restricted.

Since the definition of factor-restricted contains an universal quantification over all substitutions, it is hard to check in practice. We therefore prove that the syntactic $*$-restricted criterion implies factor-restricted.

Lemma 6. *All $*$-restricted protocol rules are factor-restricted.*

Before we can prove this lemma, we have to prove some auxiliary results about the interaction of *avars*, *factors*, and *afactors* with normalization and instantiation.

Definition 5. We define the set of *products* as

$$\mathit{Prod} = \{\mathrm{inv}^k(a * b) \mid k \in \mathbb{N} \wedge a, b \in \mathcal{T}\}.$$

Lemma 7. *For all terms t, either $t \in \mathit{Prod}$ and $\mathit{factors}(t)\big\downarrow_{DH} \subseteq_{AC} \mathit{afactors}(t)\big\downarrow_{DH}$ or $t \notin \mathit{Prod}$ and $\mathit{factors}(t)\big\downarrow_{DH} \subseteq_{AC} \{t, t^{-1}\}\big\downarrow_{DH}$. Hence $\mathit{factors}(t)\big\downarrow_{DH} \subseteq_{AC} \{t, t^{-1}\}\big\downarrow_{DH} \cup \mathit{afactors}(t)\big\downarrow_{DH}$.*

Proof: Let t' such that $t = \mathrm{inv}^k(t')$ and $\mathit{root}(t') \neq \mathrm{inv}$. We perform a case distinction on the outermost function symbol of t'. If $t' = a * b$, then $t \in \mathit{Prod}$ and $\mathit{factors}(t) = \mathit{factors}(a) \cup \mathit{factors}(b) \subseteq \mathit{afactors}(t)$. If $\mathit{root}(t') \notin \{*\}$, then $t \notin \mathit{Prod}$ and $\mathit{factors}(t)\big\downarrow_{DH} =_{AC} \{t', t'^{-1}\}\big\downarrow_{DH} =_{AC} \{t, t^{-1}\}\big\downarrow_{DH}$. ■

Lemma 8. *For all terms t and t' such that $t \rightarrow_{DH,AC} t'$, $\mathit{factors}(t')\big\downarrow_{DH} \subseteq_{AC} \mathit{factors}(t)\big\downarrow_{DH} \cup \mathit{afactors}(t)\big\downarrow_{DH} \cup \{1\}$.*

Proof: We prove this by induction over terms. First, note that the base cases for variables and names hold since no rewrite rule is applicable

- $t = inv(s)$:
  If t is rewritten below the root position, then $t' = inv(s')$ for some s' such that $s \rightarrow_{DH,AC} s'$ and we can conclude the case as follows.

$$\begin{aligned}
& \mathit{factors}(t')\big\downarrow_{DH} \\
=_{AC} \quad & [\text{since } t' = \mathrm{inv}(s')] \\
& \mathit{factors}(s')\big\downarrow_{DH} \\
\subseteq_{AC} \quad & [\text{by IH}] \\
& \mathit{factors}(s)\big\downarrow_{DH} \cup \mathit{afactors}(s)\big\downarrow_{DH} \cup \{1\} \\
=_{AC} \quad & [\text{since } t = \mathrm{inv}(s)] \\
& \mathit{factors}(t)\big\downarrow_{DH} \cup \mathit{afactors}(t)\big\downarrow_{DH} \cup \{1\}
\end{aligned}$$

  If t is rewritten at the root position, then we have to consider the rules (L1), (L6), and (L8). For all rewriting steps $t \rightarrow_{DH,AC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{factors}(t') =_{AC} \mathit{factors}(t)$.
- $t = g(s_1, \ldots, s_k)$ for $g \notin \{*, \hat{}, \mathrm{inv}\}$:
  If t is rewritten below the root position, then $\mathit{root}(t') = g$, $t, t' \notin \mathit{Prod}$, and therefore

$$\begin{aligned}
\mathit{factors}(t')\big\downarrow_{DH} &=_{AC} \{t', t'^{-1}\}\Big\downarrow_{DH} \\
&=_{AC} \{t, t^{-1}\}\Big\downarrow_{DH} \\
&=_{AC} \mathit{factors}(t)\big\downarrow_{DH}.
\end{aligned}$$

  If t is rewritten at the root position, then there is a substitution σ and a rewriting rule $l \rightarrow r$ from R_{ST} or equal to one of the rules for pairing such that $t =_{AC} l\sigma$ and $t' =_{AC} r\sigma$. We have to consider two cases. First, r is a ground term built over Σ_{ST}. Then $t, t' \notin \mathit{Prod}$ and the same reasoning as in the previous case applies. Second, r is a proper subterm of l and $t' = t|_p$ for some accessible position p. To see that p is accessible, first note that no position strictly above p is a variable position in l, l does not contain $*$, and p is a valid position in l. Hence $\mathit{root}(l|_{\tilde{p}}) = \mathit{root}(t|_{\tilde{p}}) \neq \exp$ for all positions $\tilde{p}$ strictly above p and p accessible. If $t' \in \mathit{Prod}$, then $\mathit{factors}(t')\big\downarrow_{DH} \subseteq_{AC} \mathit{afactors}(t')\big\downarrow_{DH} \subseteq_{AC} \mathit{afactors}(t)\big\downarrow_{DH}$ by Lemma 7 and because p is accessible in t. If $t' \notin \mathit{Prod}$, then $\mathit{factors}(t')\big\downarrow_{DH} =_{AC} \{t', t'^{-1}\}\big\downarrow_{DH} =_{AC} \{t, t^{-1}\}\big\downarrow_{DH} =_{AC} \mathit{factors}(t)\big\downarrow_{DH}$ since t is also not in *Prod*.
- $t = a\,\hat{}\,b$:
  There are three possibilities, either t is rewritten below the root position, at the root position with rule (D9), or at the root position with the rule (D10). For the the first two cases, and the third case with a not a product, $t' \notin \mathit{Prod}$ and

$$\begin{aligned}
\mathit{factors}(t')\big\downarrow_{DH} &=_{AC} \{t', t'^{-1}\}\Big\downarrow_{DH} \\
&=_{AC} \{t, t^{-1}\}\Big\downarrow_{DH} \\
&=_{AC} \mathit{factors}(t)\big\downarrow_{DH}.
\end{aligned}$$

For the third case with $a \in \mathit{Prod}$, then and $t' = a$ and

$$\begin{aligned} \mathit{factors}(a){\downarrow}_{DH} &\subseteq_{AC} \mathit{afactors}(a){\downarrow}_{DH} \\ &\subseteq_{AC} \mathit{afactors}(t){\downarrow}_{DH} \end{aligned}$$

since a accessible in t.

- $t = s_1 * s_2$:
If t is rewritten below the root position, then we assume without loss of generality that $t' =_{AC} s_1' * s_2$ for some s_1' with $s_1 \rightarrow_{DH,AC} s_1'$. Then we can conclude the case as follows.

$$\begin{aligned} & \mathit{factors}(t'){\downarrow}_{DH} \\ =_{AC} \quad & [\text{since } t' = s_1' * s_2] \\ & \mathit{factors}(s_1'){\downarrow}_{DH} \cup \mathit{factors}(s_2){\downarrow}_{DH} \\ \subseteq_{AC} \quad & [\text{by IH}] \\ & \mathit{factors}(s_1){\downarrow}_{DH} \cup \mathit{afactors}(s_1){\downarrow}_{DH} \\ & \cup \{1\} \cup \mathit{factors}(s_2){\downarrow}_{DH} \\ \subseteq_{AC} \quad & [\text{since } t = s_1 * s_2] \\ & \mathit{factors}(t){\downarrow}_{DH} \cup \mathit{afactors}(t){\downarrow}_{DH} \cup \{1\} \end{aligned}$$

If t is rewritten at the root position, then we have to consider the rules (L2)–(L5) and (L7), and (L9)–(L10). For all rewriting steps $t \rightarrow_{DH,AC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{factors}(t') \subseteq_{AC} \mathit{factors}(t) \cup \{1\}$.

■

Lemma 9. *For all terms t and t' such that $t \rightarrow_{DH,AC} t'$, $\mathit{afactors}(t'){\downarrow}_{DH} \subseteq_{AC} \mathit{afactors}(t){\downarrow}_{DH} \cup \{1\}$.*

Proof: We prove this by induction over terms. First, note that the base cases for variables and names hold since no rewrite rule is applicable.

- $t = \mathrm{inv}(s)$:
If t is rewritten below the root position, then $t' =_{AC} \mathrm{inv}(s')$ for some s' with $s \rightarrow_{DH,AC} s'$ and we can prove the case as follows.

$$\begin{aligned} & \mathit{afactors}(t'){\downarrow}_{DH} \\ =_{AC} \quad & [\text{since } t' = \mathrm{inv}(s')] \\ & \mathit{afactors}(s'){\downarrow}_{DH} \\ \subseteq_{AC} \quad & [\text{by IH}] \\ & \mathit{afactors}(s){\downarrow}_{DH} \cup \{1\} \\ =_{AC} \quad & [\text{since } t = \mathrm{inv}(s)] \\ & \mathit{afactors}(t){\downarrow}_{DH} \cup \{1\} \end{aligned}$$

If t is rewritten at the root position, then we have to consider the rules (L1), (L6), and (L8). For all rewriting steps $t \rightarrow_{DH,AC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{afactors}(t') =_{AC} \mathit{afactors}(t)$.

- $t = g(s_1, \ldots, s_k)$ for $g \notin \{*, \hat{}, \mathrm{inv}\}$:
If t is rewritten below the root position, then $t' = g(s_1', \ldots, s_k')$ such that $s_i \rightarrow_{DH,AC} s_i'$ for some $1 \leq i \leq k$ and $s_j =_{AC} s_j'$ for $j \neq i$ and we can prove the case as follows.

$$\begin{aligned} & \mathit{afactors}(t'){\downarrow}_{DH} \\ =_{AC} \quad & [\text{since } t' = g(s_1', \ldots, s_k')] \\ & \cup_{i=1}^{k} \mathit{afactors}(s_i'){\downarrow}_{DH} \\ \subseteq_{AC} \quad & [\text{by IH}] \\ & (\cup_{i=1}^{k} \mathit{afactors}(s_i){\downarrow}_{DH}) \cup \{1\} \\ =_{AC} \quad & [\text{since } t = g(s_1, \ldots, s_k)] \\ & \mathit{afactors}(t){\downarrow}_{DH} \cup \{1\} \end{aligned}$$

If t is rewritten at the root position, then then there is a substitution σ and a rewriting rule $l \rightarrow r$ from R_{ST} or equal to one of the rules for pairing such that $t =_{AC} l\sigma$ and $t' =_{AC} r\sigma$. We have to consider two cases. First, r is a ground term built over Σ_{ST}. Then $\mathit{afactors}(t') = \emptyset$. Second, r is a proper subterm of l and $t' =_{AC} t|_p$ for some accessible position p. Hence, for all $p' \in \mathit{appos}(t')$, there exists a $p'' \in \mathit{appos}(t)$ with $t'|_{p'} =_{AC} t|_{p''}$. Hence $\mathit{afactors}(t') \subseteq_{AC} \mathit{afactors}(t)$. To see that p is accessible, first note that there is a subterm rule $l \rightarrow r$ such that $t =_{AC} l\sigma$. Since no position strictly above p is a variable position in l and l does not contain $*$, p is a valid position in l. Hence $\mathit{root}(t|_{\tilde{p}}) = \mathit{root}(l|_{\tilde{p}}) \neq \exp$ for all positions $\tilde{p}$ strictly above p and p accessible.

- $t = a \,\hat{}\, b$:
If t is rewritten below the root position, then either $t' =_{AC} a' \hat{} b$ for some a' with $a \rightarrow_{DH,AC} a'$ or $t' = a \hat{} b'$ for some b' with $b \rightarrow_{DH,AC} b'$. The first case can be proved as follows.

$$\begin{aligned} & \mathit{afactors}(t'){\downarrow}_{DH} & \\ =_{AC} \; & \mathit{afactors}(a'){\downarrow}_{DH} & [\text{exponent not accessible}] \\ \subseteq_{AC} \; & \mathit{afactors}(a){\downarrow}_{DH} \cup \{1\} & [\text{by IH}] \\ =_{AC} \; & \mathit{afactors}(t){\downarrow}_{DH} \cup \{1\} & [\text{since } t = a \,\hat{}\, b] \end{aligned}$$

In the second case, $\mathit{afactors}(t'){\downarrow}_{DH} =_{AC} \mathit{afactors}(a){\downarrow}_{DH} =_{AC} \mathit{afactors}(t){\downarrow}_{DH}$.
If t is rewritten at the root position, then we have to consider the rules (D9) and (D10). In the first case, there are g and c such that $t = (g \,\hat{}\, c) \,\hat{}\, b$ and $t' = g \,\hat{}\, (c * b)$. We can prove this case as follows.

$$\begin{aligned} & \mathit{afactors}(t'){\downarrow}_{DH} & \\ =_{AC} \; & \mathit{afactors}(g){\downarrow}_{DH} & [\text{exponent not accessible}] \\ =_{AC} \; & \mathit{afactors}(t){\downarrow}_{DH} & [\text{since } t = (g \,\hat{}\, c) \,\hat{}\, b] \end{aligned}$$

In the second case $b = 1$, $t' =_{AC} a$ and hence

$$\begin{aligned} \mathit{afactors}(t'){\downarrow}_{DH} &=_{AC} \mathit{afactors}(a){\downarrow}_{DH} \\ &=_{AC} \mathit{afactors}(a \,\hat{}\, 1){\downarrow}_{DH}. \end{aligned}$$

- $t = s_1 * s_2$: If t is rewritten below the root position, then we assume without loss of generality that $t' = s_1' * s_2$ for some s_1' with $s_1 \rightarrow_{DH,AC} s_1'$. Then we can conclude the case as follows.

$$\begin{aligned}
& \mathit{afactors}(t')\downarrow_{DH} \\
=_{AC} \quad & [t' = s_1' * s_2] \\
& \mathit{afactors}(s_1')\downarrow_{DH} \cup \mathit{afactors}(s_2)\downarrow_{DH} \\
& \cup \mathit{factors}(s_1')\downarrow_{DH} \cup \mathit{factors}(s_2)\downarrow_{DH} \\
\subseteq_{AC} \quad & [\text{by IH}] \\
& \mathit{afactors}(s_1)\downarrow_{DH} \cup \mathit{afactors}(s_2)\downarrow_{DH} \\
& \cup \mathit{factors}(s_1')\downarrow_{DH} \cup \mathit{factors}(s_2)\downarrow_{DH} \cup \{1\} \\
\subseteq_{AC} \quad & [\text{by Lemma 8}] \\
& \mathit{afactors}(s_1)\downarrow_{DH} \cup \mathit{afactors}(s_2)\downarrow_{DH} \\
& \cup \mathit{factors}(s_1)\downarrow_{DH} \cup \mathit{factors}(s_2)\downarrow_{DH} \cup \{1\} \\
=_{AC} \quad & [\text{since } t = s_1 * s_2] \\
& \mathit{afactors}(t)\downarrow_{DH} \cup \{1\}
\end{aligned}$$

If t is rewritten at the root position, then we have to consider the rules (L2)–(L5), (L7), (L9)–(L10). For all rewriting steps $t \rightarrow_{DH,AC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{afactors}(t) \subseteq_{AC} \mathit{afactors}(t') \cup \{1\}$.

■

Lemma 10. *For all terms t,*

$$\mathit{afactors}(\,t\downarrow_{DH}) \subseteq_{AC} \mathit{afactors}(t)\downarrow_{DH}.$$

Proof: Directly follows from t $\downarrow_{DH}$-normal or $t \rightarrow^{+}_{DH,AC} t\downarrow_{DH}$ using Lemma 9 and the fact that for a product s that is $\downarrow_{DH}$-normal, $1 \notin \mathit{factors}(s)$. ■

Lemma 11. *For all terms t,*

$$\mathit{factors}(\,t\downarrow_{DH}) \subseteq_{AC} \mathit{factors}(t)\downarrow_{DH} \cup \mathit{afactors}(t)\downarrow_{DH} \cup \{1\}.$$

Proof: We prove this by induction over the length of the rewriting sequence $t_1 \rightarrow \ldots \rightarrow t_k$ where t_k $\downarrow_{DH}$-normal. The base case clearly holds and assuming the hypothesis for $k-1$, we conclude with

$$\begin{aligned}
& \mathit{factors}(t_k) \\
\subseteq_{AC} \quad & [\text{IH}] \\
& \mathit{factors}(t_2)\downarrow_{DH} \cup \mathit{afactors}(t_2)\downarrow_{DH} \cup \{1\} \\
\subseteq_{AC} \quad & [\text{Lemma 8 and 9}] \\
& \mathit{factors}(t_1)\downarrow_{DH} \cup \mathit{afactors}(t_1)\downarrow_{DH} \cup \{1\}.
\end{aligned}$$

■

Lemma 12. *For all terms t and $\downarrow_{DH}$-normal substitutions σ,*

$$\begin{aligned}\mathit{afactors}(t\sigma)\downarrow_{DH} =_{AC}\ & (\mathit{afactors}(t)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH} \\ & \cup (\cup_{x \in \mathit{avars}(t)}\ \mathit{afactors}(x\sigma)\downarrow_{DH}).\end{aligned}$$

Proof: Let σ arbitrary and

$$F_t = (\mathit{afactors}(t)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH} \cup (\cup_{x \in \mathit{avars}(t)}\ \mathit{afactors}(x\sigma)\downarrow_{DH}).$$

Note that

$$\begin{aligned}
& \cup_{i=1}^{k} \mathit{afactors}(s_i) =_{AC} \mathit{afactors}(t) \text{ and} \\
& \cup_{i=1}^{k} \mathit{avars}(s_i) = \mathit{avars}(t) \\
\text{implies } & \cup_{i=1}^{k} F_{s_i} =_{AC} F_t. \qquad (1)
\end{aligned}$$

We prove $\mathit{afactors}(t\sigma)\downarrow_{DH} =_{AC} F_t$ by induction on t.

- $t = x$ for $x \in \mathcal{V}_{msg}$: Since $\mathit{afactors}(x) = \emptyset$ and $\mathit{avars}(x) = \{x\}$, we have $F_x = \mathit{afactors}(x\sigma)\downarrow_{DH}$.
- $t = x$ for $x \in \mathcal{V}_{pub} \cup \mathcal{V}_{fresh}$: Both sides are equal to the empty set since σ well-sorted.
- $t \in PN \cup FN$: Both sides are equal to the empty set.
- $t = \mathrm{inv}(s)$:

$$\begin{aligned}\mathit{afactors}(t\sigma)\downarrow_{DH} &=_{AC} \mathit{afactors}(s\sigma)\downarrow_{DH} \\ &=_{AC} F_s \\ &=_{AC} F_t\end{aligned}$$

  using the induction hypothesis and (1).
- $t = g(s_1, \ldots, s_k)$ for $g \notin \{*, \mathrm{inv}, expo\}$: Using the induction hypothesis and (1), we obtain

$$\begin{aligned}\mathit{afactors}(t\sigma)\downarrow_{DH} &=_{AC} (\cup_{i=1}^{k} \mathit{afactors}(s_i\sigma)\downarrow_{DH}) \\ &=_{AC} (\cup_{i=1}^{k} F_{s_i}) \\ &=_{AC} F_t.\end{aligned}$$

- $t = s_1 \,\hat{}\, s_2$: Using the induction hypothesis and (1), we obtain

$$\begin{aligned}\mathit{afactors}(t\sigma)\downarrow_{DH} &=_{AC} \mathit{afactors}(s_1\sigma)\downarrow_{DH} \\ &=_{AC} F_{s_1} \\ &=_{AC} F_t.\end{aligned}$$

- $t = s_1 * s_2$: Let $J \subseteq \{1,2\}$ such that $j \in J$ iff $s_j\sigma \notin \mathit{Prod}$. We first show that

$$F_{s_1} \cup F_{s_2} \cup (\cup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH}) =_{AC} F_t. \quad (2)$$

  - "$\subseteq$": If $J = \emptyset$, then we are done since $\mathit{avars}(s_1) \cup \mathit{avars}(s_2) \subseteq \mathit{avars}(t)$ and $\mathit{afactors}(s_1) \cup \mathit{afactors}(s_s) \subseteq \mathit{afactors}(t)$ and therefore $F_{s_1} \cup F_{s_2} \subseteq F_t$. Now let $j \in J$ arbitrary. Then $s_j\sigma \notin \mathit{Prod}$ and hence $s_j \notin \mathit{Prod}$. Then there are k and u such that $s_j = \mathrm{inv}^k(u)$ and $\mathit{root}(u) \notin \{*, \mathrm{inv}\}$. Note that $u\sigma \notin \mathit{Prod}$ and $u^{-1}\sigma \notin \mathit{Prod}$. Therefore $\{u, u^{-1}\} = \mathit{factors}(s_j) \subseteq \mathit{afactors}(t)$ and

$$\begin{aligned}\{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH} &=_{AC} \{u\sigma, u^{-1}\sigma\}\downarrow_{DH} \\ &\subseteq_{AC} (\mathit{afactors}(t)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH} \\ &\subseteq_{AC} F_t\ .\end{aligned}$$

– "$\supseteq$": Since

$$\begin{aligned}F_t =_{AC}\; & (\mathit{afactors}(t)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}\\ & \cup (\cup_{x\in \mathit{avars}(t)}\; \mathit{afactors}(x\sigma)\downarrow_{DH})\\ =_{AC}\; & (\cup_{i=1}^{k}\; (\mathit{afactors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH})\\ & \cup (\cup_{i=1}^{k}\; (\mathit{factors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH})\\ & \cup (\cup_{i=1}^{k}(\cup_{x\in \mathit{avars}(t)}\; \mathit{afactors}(x\sigma)\downarrow_{DH}))\end{aligned}$$

we have to show that for $i \in \{1,2\}$,

$$\begin{aligned}&(\mathit{factors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH})\\ \subseteq_{AC}\; & F_{s_1} \cup F_{s_2} \cup (\cup_{j\in J}\; \{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH})\ .\end{aligned}$$

Let $i \in \{1,2\}$ arbitrary. We distinguish two cases. First, if $s_i \in \mathit{Prod}$, then $\mathit{factors}(s_i) \subseteq \mathit{afactors}(s_i)$ and hence

$$(\mathit{factors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH} \subseteq_{AC} F_{s_i}.$$

Second, if $s_i \notin \mathit{Prod}$, then there are k and u such that $s_i = \mathrm{inv}^k(u)$ and $\mathit{root}(u) \notin \{*, \mathrm{inv}\}$. Hence $\mathit{factors}(s_i) = \{u, u^{-1}\}$ and either $u\sigma \in \mathit{Prod}$ or not. If $u\sigma \in \mathit{Prod}$, then $u^{-1}\sigma \in \mathit{Prod}$ and hence $(\mathit{factors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}) = \emptyset$. If $u\sigma \notin \mathit{Prod}$, then $s_i\sigma \notin \mathit{Prod}$ and hence $i \in J$. Then we can conclude the case with

$$\begin{aligned}&(\mathit{factors}(s_i)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}\\ =_{AC}\; & \{u\sigma, u^{-1}\sigma\}\downarrow_{DH}\\ =_{AC}\; & \{s_i\sigma, (s_i\sigma)^{-1}\}\downarrow_{DH}\\ \subseteq_{AC}\; & F_{s_1} \cup F_{s_2} \cup (\cup_{j\in J}\; \{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH})\ .\end{aligned}$$

Using the previous result we can now conclude the proof as follows.

$$\begin{aligned}& \mathit{afactors}(t\sigma)\downarrow_{DH}\\ =_{AC}\quad & [t = s_1 * s_2]\\ & \cup_{i=1}^{2}\,(\mathit{afactors}(s_i\sigma)\downarrow_{DH} \cup \mathit{factors}(s_i\sigma)\downarrow_{DH})\\ \subseteq_{AC}\quad & [\text{by Lemma 7}]\\ & (\cup_{i=1}^{2}\; \mathit{afactors}(s_i\sigma)\downarrow_{DH})\\ & \cup (\cup_{j\in J}\; \{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH})\\ \subseteq_{AC}\quad & [\text{by IH}]\\ & F_{s_1} \cup F_{s_2} \cup (\cup_{j\in J}\; \{s_j\sigma, (s_j\sigma)^{-1}\}\downarrow_{DH})\\ \subseteq_{AC}\quad & [(2)]\\ & F_t\end{aligned}$$

■

Proof of Lemma 6: Let $l\text{-}[\, a \,]\!\to r$ be an arbitrary $*$-restricted multiset rewriting rule and σ an arbitrary $\downarrow_{DH}$-normal substitution. Then the following holds and $l\text{-}[\, a \,]\!\to r$ is therefore factor-restricted.

$$\begin{aligned}& \mathit{afactors}(\,(r\sigma)\downarrow_{DH})\\ \subseteq_{AC}\quad & [\text{Lemma 10}]\\ & \mathit{afactors}(r\sigma)\downarrow_{DH}\\ =_{AC}\quad & [\text{Lemma 12}]\\ & (\mathit{afactors}(r)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}\\ & \cup (\cup_{x\in \mathit{avars}(r)}\; \mathit{afactors}(x\sigma)\downarrow_{DH})\\ \subseteq_{AC}\quad & [\text{no } * \text{ in } l\text{-}[\, a \,]\!\to r]\\ & (\mathit{afactors}(l)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}\\ & \cup (\cup_{x\in \mathit{avars}(r)}\; \mathit{afactors}(x\sigma)\downarrow_{DH})\\ \subseteq_{AC}\quad & [(*)]\\ & (\mathit{afactors}(l)\sigma \smallsetminus \mathit{Prod})\downarrow_{DH}\\ & \cup (\cup_{x\in \mathit{avars}(l)}\; \mathit{afactors}(x\sigma)\downarrow_{DH})\\ =_{AC}\quad & [\text{Lemma 12}]\\ & \mathit{afactors}(l\sigma)\downarrow_{DH}\\ =_{AC}\quad & [\,(l\sigma)\downarrow_{DH} =_{AC} l\sigma]\\ & \mathit{afactors}(\,(l\sigma)\downarrow_{DH})\end{aligned}$$

(*) Since $l\text{-}[\, a \,]\!\to r$ is a protocol rule, $\mathit{vars}(r) \subseteq \mathit{vars}(l) \cup \mathcal{V}_{pub}$. Hence $\mathit{avars}(r) \subseteq \mathit{vars}(l) \cap \mathcal{V}_{msg} = \mathit{avars}(l)$ since l does not contain exp and all variables are accessible. ■

2) Normal Message Deduction: To obtain the set of normal message deduction rules *ND* that accounts for Σ_{ST} and R_{ST}, we proceed as follows. We remove the construction rules for enc, dec, and h. We remove the deconstruction rule for dec. Then, we add construction rules for all the function symbols in Σ_{ST}. Finally, for each rewrite rule $l \to r$ such that there is a position p with $r = l|_p$, we compute the set of destruction rules.

We use the following function to compute the rules, ignoring the exponentiation tags, which we will add later.

$$\begin{aligned}&\mathit{decon\text{-}rules}(l,p) =\\ &\quad \{[\mathsf{K}^{\downarrow}(l|_{p'})] \cdot \mathit{con\text{-}prems}(l,p')\text{-}[\,]\!\to[\mathsf{K}^{\downarrow}(l|_p)]\\ &\quad \mid p' \text{ strictly above } p \text{ and } p' \neq [\,]\}\end{aligned}$$

The *con-prems* function computes the corresponding $\mathsf{K}^{\uparrow}$-premises and is defined as folllows.

$$\begin{aligned}&\mathit{con\text{-}prems}(l,p) =\\ &\quad \mathit{seq}(\{\mathsf{K}^{\uparrow}(l|_{p'}) \mid p' \neq [\,] \wedge \exists p''.\, p'' \text{ above or equal to } p\\ &\qquad\qquad \text{and } p'' \text{ sibling of } p'\})\end{aligned}$$

Here, $\mathit{seq}(S)$ denotes a sequence that consists of the elements of the set S. We annotate the rules with exponentiation tags by using exp for the conclusions and using all possible combinations of exp and noexp for the premises. For rewrite rules where r is ground, we do not require deconstruction rules since such an r can be constructed directly by using the correponding construction rules.

Example 9. Consider the rule $\mathrm{dec}(\mathrm{enc}(x,y),y) \to x$. We compute the set of deconstruction rules as follows. First we compute *decon-rules*$(\mathrm{dec}(\mathrm{enc}(x,y),y),[1,1])$. The only position $p' \neq []$ strictly above $[1,1]$ is $[1]$. So the only $\mathsf{K}^{\downarrow}$-premise that we have to consider is $\mathsf{K}^{\downarrow}(\mathrm{enc}(x,y))$. Hence the result is

$$\{[\mathsf{K}^{\downarrow}(\mathrm{enc}(x,y))] \cdot [\mathsf{K}^{\uparrow}(y)] \text{-}[]\!\!\rightarrow [\mathsf{K}^{\downarrow}(x)]\}$$

since *con-prems*$(\mathrm{dec}(\mathrm{enc}(x,y),y),[1]) = [\mathsf{K}^{\uparrow}(y)]$.

Consider the rule $a(b(c(x_1,x_2),x_3),x_4) \to x_1$. Then $p = [1,1,1]$ and the choices for p' are $[1]$ and $[1,1]$. The first choice results in the rule

$$[\mathsf{K}^{\downarrow}(b(c(x_1,x_2),x_3))] \cdot [\mathsf{K}^{\uparrow}(x_4)] \text{-}[]\!\!\rightarrow [\mathsf{K}^{\downarrow}(x_1)]$$

which corresponds to the adversary applying a to $b(c(x_1,x_2),x_3)$ and x_4 and thereby obtaining x_1. The position $p' = [1,1]$ results in

$$[\mathsf{K}^{\downarrow}(c(x_1,x_2))] \cdot [\mathsf{K}^{\uparrow}(x_3), \mathsf{K}^{\uparrow}(x_4)] \text{-}[]\!\!\rightarrow [\mathsf{K}^{\downarrow}(x_1)].$$

which corresponds to the adversary first applying b to $c(x_1,x_2)$ and x_3 and the applying a to the result and x_4. The result after rewriting is x_1 as before. These two rules cover all the possibilities to apply the function symbol a to generate a term that can be reduced with $a(b(c(x_1,x_2),x_3),x_4) \to x_1$ at the topmost position.

3) Normal Dependency Graphs: To prove Lemma 1, we require a few more lemmas and definitions. In the rest of this section P is always assumed to be $*$-restricted.

Definition 6. We define the *known messages* of a dependency graph *dg* as

$$\mathit{known}(dg) = \{m \mid \textit{exists conclusion fact } \mathsf{K}(m) \textit{ in } dg\}.$$

We define the *up-known (resp. down-known) messages* of a normal dependency graph *ndg* as

$$\mathit{known}^{d}(ndg) = \{m \mid \textit{exists conclusion fact } \mathsf{K}^{d}_{f}(m) \textit{ in } dg\}$$

for $d = \uparrow$ (resp. $d = \downarrow$). We define the *known messages* for a normal dependency graph *ndg* as

$$\mathit{known}\updownarrow(ndg) = \mathit{known}\uparrow(ndg) \cup \mathit{known}\downarrow(ndg).$$

We define the *available state-conclusions of a dependency graph dg* as

$$\mathit{stfacts}(dg) = \{f \mid f \in \mathit{cfacts}(dg) \wedge (\forall m\ d\ e.\ f \neq \mathsf{K}(m) \wedge f \neq \mathsf{K}^{d}_{e}(m))\}^{\sharp},$$

where *cfacts*(dg) denotes the consumable facts in *dg*. We define the *created messages* of a dependency graph *dg* as

$$\mathit{created}(dg) = \{n \mid \textit{exists conclusion fact } \mathsf{Fr}(n) \textit{ in } dg\}.$$

A normal dependency graph $ndg' = (I',D')$ is a *deduction extension* of $ndg = (I,D)$ if I is a prefix of I', $D \subseteq D'$, *trace*(ndg') = *trace*(ndg), *stfacts*(ndg) = *stfacts*(ndg'), and *created*(ndg) = *created*(ndg'). If there is a deduction extension such that a message m is known, we write m *is deducible*.

In the following, we use $\mathrm{F}\downarrow$, respectively $\mathrm{F}\uparrow$ to denote the construction and destruction rules for the corresponding function symbols. For example, we use $\mathrm{FST}\downarrow$ to denote the destruction rule for fst.

Lemma 13. *For all ndg* $\in$ *ndgraphs*(P)*, conclusions* (i,u) *in ndg with conclusion fact* f *and* $t \in_{AC}$ *afactors*(f)*, there is a conclusion* (j,v) *with conclusion fact* $\mathsf{K}^{d}_{e}(m)$ *in ndg with* $j < i$ *and* $m \in_{AC} \{t, (t^{-1})\downarrow_{DH}\}$.

Proof: We prove by induction on normal dependency graphs that the property holds. The property obviously holds for $([],\varnothing)$. Let $ndg = (I,D) \in \mathit{ndgraphs}(P)$ arbitrary, $l \text{-}[\ a\ ]\!\!\rightarrow r \in \mathit{ginsts}(\lceil P \rceil^{DH} \cup ND \cup \{\textsc{Fresh}\})$, $l \text{-}[\ a\ ]\!\!\rightarrow r\ \downarrow_{DH}$-normal, and D' such that $ndg' = (I \cdot [l \text{-}[\ a\ ]\!\!\rightarrow r], D \uplus D') \in \mathit{ndgraphs}(P)$. We perform a case distinction on $l \text{-}[\ a\ ]\!\!\rightarrow r$.

- If $l \text{-}[\ a\ ]\!\!\rightarrow r \in \mathit{ginsts}(\textsc{Fresh})$, then there is nothing to show since $f = \mathsf{Fr}(n)$ for a fresh name n and therefore *afactors*$(f) = \varnothing$.
- If $l \text{-}[\ a\ ]\!\!\rightarrow r \in \mathit{ginsts}(\lceil P \rceil^{DH})$, then there is a substitution σ that is grounding for some $l' \text{-}[\ a'\ ]\!\!\rightarrow r'$ in P such that $l \text{-}[\ a\ ]\!\!\rightarrow r =_{AC} (l' \text{-}[\ a'\ ]\!\!\rightarrow r')\sigma\downarrow_{DH}$. Since P is $*$-restricted and therefore factor-restricted,

$$\begin{aligned}\mathit{afactors}(r) &=_{AC} \mathit{afactors}(\,(r'\sigma)\downarrow_{DH})\\ &\subseteq_{AC} \mathit{afactors}(\,(l'\sigma)\downarrow_{DH}) =_{AC} \mathit{afactors}(l).\end{aligned}$$

  Hence, for all $j \in \mathit{idx}(r)$ and $t \in_{AC} \mathit{afactors}(r_j)$, there is $k \in \mathit{idx}(l)$ such that $t \in_{AC} \mathit{afactors}(l_k)$. Because the dependency must be satisfied by some conclusion in *ndg*, there is a conclusion c with a conclusion fact that is equal to l_k modulo AC. We can therefore use the induction hypothesis.
- If $l \text{-}[\ a\ ]\!\!\rightarrow r \in \mathit{ginsts}(ND)$, then all rules except the multiplication rule, $\textsc{Isend}$, $\textsc{Coerce}$, $\textsc{Fresh}\uparrow$, and $\textsc{Irecv}$ are of the form $F_1(m_1),\ldots,F_k(m_k) \text{-}[]\!\!\rightarrow F(\,f(m_1,\ldots,m_k)\downarrow_{DH})$ with $f \neq *$ and m_i in normal form. We can use Lemma 10 to obtain

$$\begin{aligned}&\mathit{afactors}(\,f(m_1,\ldots,m_k)\downarrow_{DH})\\ \subseteq_{AC}\ &\mathit{afactors}(f(m_1,\ldots,m_k))\downarrow_{DH}\\ \subseteq_{AC}\ &\bigcup_{i=1}^{k} \mathit{afactors}(m_i)\end{aligned}$$

  and use the induction hypothesis. For the multiplication case, note that the only new *afactors* are the inputs and their inverses and we can therefore use the induction hypothesis too. For $\textsc{Isend}$, $\textsc{Coerce}$, and $\textsc{Irecv}$, note that *afactors*$(l) =_{AC}$ *afactors*(r) and we can therefore use the induction hypothesis.

■

Lemma 14. *For all ndg $\in$ ndgraphs(P) and conclusion facts $\mathsf{K}^{\downarrow}_{e}(m)$ in ndg, there is a deduction extension ndg$'$ that contains a conclusion fact $\mathsf{K}^{\uparrow}_{e'}(m')$ with $m =_{AC} m'$. Furthermore, if $e = \mathsf{exp}$ then $e' = \mathsf{exp}$.*

Proof: We have to consider two cases. First, if there is a conclusion c' with conclusion fact $\mathsf{K}^{\uparrow}_{e'}(m')$ in *ndg* such that $m =_{AC} m'$, then we do not have to extend *ndg*. Because of **N6**, we know that c' must be the conclusion of an instance of COERCE or the construction rule for pairing or inversion. In both cases $e' = \mathsf{exp}$. If there is no such conclusion c' in *ndg*, we use induction on m.

- If m is not a pair or inverse, we can use COERCE directly and $e = e'$. Note that by **N4**, m is not a product.
- If $m = \langle u, v\rangle$, then we can obtain a deduction extension of *ndg* that contains $\mathsf{K}^{d_1}_{e_1}(u)$ and $\mathsf{K}^{d_2}_{e_2}(v)$ by applying FST$\downarrow$ and SND$\downarrow$ if required. If $d_1 = \downarrow$ or $d_2 = \downarrow$, we can use the induction hypothesis to deduce $\mathsf{K}^{\uparrow}_{e_3}(u)$ and $\mathsf{K}^{\uparrow}_{e_4}(v)$. Afterwards, we can use PAIR$\uparrow$ to deduce $\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\langle u, v\rangle)$ as required.
- The case for $a = u^{-1}$ is analogous.

Note that these extensions do not violate **N6**. If there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\downarrow}_{e'}(m)$ and we add a node with conclusion $(j, 1)$ with fact $\mathsf{K}^{\uparrow}_{e}(m')$ for $m =_{AC} m'$, then j is an instance of PAIR$\uparrow$, INV$\uparrow$, or COERCE, and $i < j$. ■

Lemma 15. *Let ndg $\in$ ndgraphs(P), $m \in$ known$\updownarrow$(ndg), m $\downarrow_{DH}$-normal, and $m \notin$ Prod, then there is a deduction extension ndg$'$ of ndg with $(m^{-1})\downarrow_{DH} \in_{AC}$ known$\updownarrow$(ndg$'$).*

Proof: We distinguish whether m is an inverse or not. If m is an inverse, then there is t such that $m = t^{-1}$ and t is not a product. Hence, $(m^{-1})\downarrow_{DH} =_{AC} t$ and $t^{-1} \in$ *known*$\downarrow$(*ndg*) or $t^{-1} \in$ *known*$\uparrow$(*ndg*). In the first case, we can use INV$\downarrow$ to deduce t since t is not a product. In the second case, t^{-1} must be the conclusion of an INV$\uparrow$ rule, which implies that $t \in$ *known*$\updownarrow$(*ndg*).

If m is no inverse, then $(m^{-1})\downarrow_{DH} =_{AC} m^{-1}$, as m is no product. Due to Lemma 14, there is an extension *ndg*$'$ of *ndg* such that $m \in$ *known*$\uparrow$(*ndg*$'$). Hence, we can extend *ndg*$'$ with INV$\uparrow$ to deduce m^{-1}. ■

Lemma 16. *Let ndg $\in$ ndgraphs(P), t $\downarrow_{DH}$-normal, and for all $m \in_{AC}$ factors(t), $m \in_{AC}$ known$\updownarrow$(ndg) or $m^{-1}\downarrow_{DH} \in$ known$\updownarrow$(ndg), then there is a deduction extension ndg$'$ of ndg with $t \in_{AC}$ known$\updownarrow$(ndg$'$).*

Proof: Proof by induction over t.

- $t \in FN \cup PN$: Since *factors*$(t) = \{t, t^{-1}\}$, t or t^{-1} is known. In the first case, we are done. In the second case, we can use Lemma 15 to obtain *ndg*$'$ where t is known since t^{-1} is in normal form and not in *Prod.*
- $t = f(u_1, \ldots, u_k)$ for $f \notin \{\text{inv}, *\}$: Since
$$\begin{aligned}&\mathit{factors}(f(u_1,\ldots,u_k))\\ &= \{f(u_1,\ldots,u_k), f(u_1,\ldots,u_k)^{-1}\},\end{aligned}$$
the reasoning is the same as in the previous case.
- $t = u^{-1}$: Since *factors*(u^{-1}) = *factors*(u), there is a deduction extension *ndg*$'$ of *ndg* such that u is known by the induction hypothesis. We can therefore use Lemma 14 to deduce $\mathsf{K}^{\uparrow}(u)$ (if required) and then deduce u^{-1} by applying INV$\uparrow$.
- $t = (u_1 * \ldots * u_k) * (u_{k+1} * \ldots * u_{k+l})^{-1}$: Since *factors*$((u_1 * \ldots * u_k) * (u_{k+1} * \ldots * u_{k+l})^{-1}) = \{u_i \mid 1 \le i \le k + l\}$, either u_i or u_i^{-1} is known in *ndg* for all i. If only u_i^{-1} is known, we can deduce u_i by Lemma 15. Hence, we can apply the $k + l$-ary multiplication rule to $u_1, \ldots, u_{k+l}$ to deduce t.

■

Lemma 17. *For all ndg $\in$ ndgraphs(P) and $s, t \in$ known$\updownarrow$(ndg), there is a deduction extension ndg$'$ of ndg with $(s * t)\downarrow_{DH} \in$ known$\updownarrow$(ndg$'$).*

Proof: By Lemma 16, it is sufficient to show that for all $m \in$ *factors*$((s * t)\downarrow_{DH})$, there is a deduction extension of *ndg* where m or $(m^{-1})\downarrow_{DH}$ known. First, note that the following holds for *factors*$((s * t)\downarrow_{DH})$.

$$\begin{aligned}
&\mathit{factors}((s * t)\downarrow_{DH})\\
\subseteq_{AC}\quad &[\text{Lemma 11}]\\
&\{1\} \cup \mathit{factors}(s * t)\downarrow_{DH} \cup \mathit{afactors}(s * t)\downarrow_{DH}\\
\subseteq_{AC}\quad &[\text{simplify}]\\
&\{1\} \cup \mathit{factors}(s) \cup \mathit{factors}(t) \cup \mathit{afactors}(s) \cup \mathit{afactors}(t)\\
\subseteq_{AC}\quad &[\text{Lemma 7}]\\
&\{1\} \cup \{s, s^{-1}\big\downarrow_{DH}\} \cup \{t, t^{-1}\big\downarrow_{DH}\}\\
&\cup\, \mathit{afactors}(s) \cup \mathit{afactors}(t)
\end{aligned}$$

Since 1, s, $s^{-1}\downarrow_{DH}$, t, $t^{-1}\downarrow_{DH}$, *afactors*(s), and *afactors*(t) are already known or deducible, this means all elements of *factors*$((s * t)\downarrow_{DH})$ are deducible. ■

We now prove a lemma that will be used to show that *decon-rules* includes all required deconstruction rules for the rewrite rules in R_{ST}.

Lemma 18. *For all ndg $\in$ ndgraphs(P), $t \in$ known$\updownarrow$(ndg), and valid positions p in t such that root$(t|_{p'}) \neq *$ for all p' above or equal to p, either*

(a) there is a position $\tilde{p}$ strictly above p such that $t|_{\tilde{p}} \in_{AC}$ known$\downarrow$(ndg) and $t|_{p'} \in_{AC}$ known$\updownarrow$(ndg) for all valid positions p' in t such that there is p'' above or equal to $\tilde{p}$ and p' sibling of p'', or

(b) $t|_p \in_{AC}$ known$\updownarrow$(ndg) and $t|_{p'} \in_{AC}$ known$\updownarrow$(ndg) for all valid positions p' in t such that there is p'' above or equal to p and p' sibling of p''.

Proof: Let *ndg* and t arbitrary. We prove the statement by induction over positions p such that *root*$(t|_{p'}) \neq *$ for all p' above or equal to p. For the empty position $[]$, (b) clearly holds since $t|_{[]} = t$ and $[]$ has no siblings. For the induction step, we first assume that (a) holds for the position p. Then (b) also holds for all valid positions $p' = p \cdot [i]$ in t. Now assume that (b) holds for p. If $t|_p$ is up-known and not down-known, then it cannot be the conclusion of the COERCE rule and must be the conclusion of a construction rule. Then $t|_p$ is either the conclusion of a multiplication rule or a construction rule of the form $\mathsf{K}^{\uparrow}_{e_1}(s_1), \ldots, \mathsf{K}^{\uparrow}_{e_k}(s_k) \text{-}[]\!\!\rightarrow \mathsf{K}^{\uparrow}_{e}(f(s_1, \ldots, s_k))$. In the first case, $\mathit{root}(t|_p) = *$ which contradicts our assumptions. In the second case, (b) also holds for all $p' = p \cdot [i]$ with $1 \leq i \leq k$ because $t|_{p \cdot [i]} = s_i \in_{AC} \mathit{known}\updownarrow(\mathit{ndg})$ and the terms at sibling positions of p' are also known. If $t|_p$ is down-known and p satisfies (b), then for all valid positions $p' = p \cdot [i]$ in t, p' satisfies (a). ■

Proof of Lemma 1:

Using Lemma 4 and Lemma 5, it suffices to show

$$\begin{aligned} &\{\, \overline{\mathit{trace}}(\mathit{dg}) \\ &\mid \mathit{dg} \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH}) \land \mathit{dg}\ {\downarrow_{DH}}\text{-normal}\} \\ =_{AC}\ &\{\, \overline{\mathit{trace}}(\mathit{ndg}) \mid \mathit{ndg} \in \mathit{ndgraphs}(P)\}. \end{aligned}$$

We prove both inclusions separately.

$\subseteq_{AC}$ **:** We show by induction over dependency graphs that for all $\mathit{dg} \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH})$ with $\mathit{dg}\ {\downarrow_{DH}}$-normal, there is $\mathit{ndg} \in \mathit{ndgraphs}(P)$ such that

$$\mathit{known}(\mathit{dg}) \subseteq_{AC} \mathit{known}\updownarrow(\mathit{ndg}) \quad (1)$$

$$\mathit{stfacts}(\mathit{dg}) \subseteq^{\sharp}_{AC} \mathit{stfacts}(\mathit{ndg}) \quad (2)$$

$$\mathit{created}(\mathit{dg}) =_{AC} \mathit{created}(\mathit{ndg}) \quad (3)$$

$$\overline{\mathit{trace}}(\mathit{dg}) =_{AC} \overline{\mathit{trace}}(\mathit{ndg}). \quad (4)$$

This clearly holds for $([], \varnothing)$. Let $\mathit{dg} = (I, D) \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH})$ with $\mathit{dg}\ {\downarrow_{DH}}$-normal, and $\mathit{ndg} = (\tilde{I}, \tilde{D}) \in \mathit{ndgraphs}(P)$ such that (1)–(4) hold. Let $ri \in \mathit{ginsts}(\lceil P \cup \mathit{MD} \rceil^{DH} \cup \{\text{FRESH}\})$ arbitrary such that $ri\ {\downarrow_{DH}}$-normal and D' arbitrary such that $\mathit{dg}' = (I \cdot [ri], D \uplus D') \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH})$. Then we have to show that there is an $\mathit{ndg}' \in \mathit{ndgraphs}(P)$ that satisfies (1)–(4) with respect to dg'.

We perform a case distinction on ri.

- $\mathit{ginsts}(\text{FRESH})$: Condition (3) for dg and ndg ensures that we can extend $\tilde{I}$ with ri to obtain ndg' without violating unique FRESH instances (**DG4**).
- $\mathit{ginsts}(\lceil P \rceil^{DH})$: We append ri to $\tilde{I}$ and extend $\tilde{D}$ with the required dependencies to obtain ndg' which is possible because of condition (2) for dg and ndg.
- $\mathit{ginsts}(\lceil \mathit{MD} \rceil^{DH})$: Let $ri = l \text{-}[\, a \,]\!\!\rightarrow r$. For all rules except for the adversary receive, send, fresh name, and public name rules, we have $l = \mathsf{K}(m_1), \ldots, \mathsf{K}(m_n)$, $a = []$, and $r = \mathsf{K}(m)$. If $m = f(m_1, \ldots, m_n)$, then we say the instance is a trivial variant of the rule. Otherwise it is nontrivial. We must show that there is a deduction extension ndg' of ndg such that $m \in_{AC} \mathit{known}\updownarrow(\mathit{ndg}')$. We can assume that $m \notin_{AC} \mathit{known}\updownarrow(\mathit{ndg})$ and $m_i \in_{AC} \mathit{known}\uparrow(\mathit{ndg})$ for $1 \leq i \leq n$ because of Lemma 14. If m is a product, then we just have to show that m is accessible in one of the premise facts. Then we can use use Lemma 13 to show that all *factors* or their inverses are known. Hence m is deducible by Lemma 16.

  - $\mathsf{Out}(m) \text{-}[]\!\!\rightarrow \mathsf{K}(m)$: We can use IRECV if m is not a product. Since m is accessible in $\mathsf{Out}(m)$, the product case is also handled.
  - $\mathsf{K}(m) \text{-}[\, \mathsf{K}(m) \,]\!\!\rightarrow \mathsf{In}(m)$: We can use ISEND.
  - $\mathsf{Fr}(n) \text{-}[]\!\!\rightarrow \mathsf{K}(n)$: We can use the construction rule for fresh names.
  - Message deduction rules for 1 and public names: We can use the construction rules for 1 and public names.
  - Trivial variant of message deduction rule for fst, snd, pair, inv, or a function symbol from Σ_{ST}: The corresponding instance of $F\uparrow$ can be used since the inputs are in $\mathit{known}\uparrow(\mathit{ndg})$.
  - Nontrivial variant of fst or snd: We have $m_1 = \langle m, m' \rangle$ for some m'. If m is a product, then m is accessible in m_1 and hence deducible. If $\langle m, m' \rangle \in \mathit{known}\downarrow(\mathit{ndg})$, then we can use the FST$\downarrow$ rule. If $\langle m, m' \rangle$ only in $\mathit{known}\uparrow(\mathit{ndg})$, then it must be the conclusion of a PAIR$\uparrow$ instance. But then, m is known since it is the first premise. The reasoning for snd is analogous to the previous case.
  - Nontrivial variant of inv: Since m_1 is known, $m = (m_1^{-1}){\downarrow_{DH}}$ is deducible by Lemma 15.
  - There are two different types of nontrivial variants of the rules for Σ_{ST}. First, for function symbols f and rewriting rules $(f(t_1, \ldots, t_k) \to a) \in R_{ST}$ such that a is a ground term in normal form, the variant is $\mathsf{K}(t_1), \ldots, \mathsf{K}(t_k) \text{-}[]\!\!\rightarrow \mathsf{K}(a)$. The message a can be deduced using the construction rules for Σ_{ST}, reusing known messages where possible.
    Second, for function symbols f and rewriting rules $(f(t_1, \ldots, t_k) \to t_i|_p) \in R_{ST}$, the variant is $\mathsf{K}(t_1), \ldots, \mathsf{K}(t_k) \text{-}[]\!\!\rightarrow \mathsf{K}(t_i|_p)$. We now consider an arbitrary instance $\mathsf{K}(m_1), \ldots, \mathsf{K}(m_k) \text{-}[]\!\!\rightarrow \mathsf{K}(m_i|_p)$ of such a rule. Then m_1 to m_k are known in ndg. If $m_i|_p$ is a product, then it is accessible in m_i and therefore deducible. If it is not a product, then p is a position in m_i such that $\mathit{root}(t|_{p'}) \neq *$ for all p' above or equal to p. We can therefore apply Lemma 18 to obtain (a) or (b) for ndg, m_i, and p. In case (b), $m_i|_p$ is already known. In case (a), there is a destruction rule from $\mathit{decon\text{-}rules}(f(t_1, \ldots, t_k), p)$ that can be applied to obtain $m_i|p$.
  - Trivial variant of exp: The only problematic case occurs if for m_1, there is only a $\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(m_1)$ fact which cannot be used as first input of the EXP$\uparrow$-rule. In this case, there must be either an EXP$\uparrow$ instance with conclusion $\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(m_1)$ or an EXP$\downarrow$

instance with conclusion $\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(m_1)$ followed by a COERCE instance with conclusion $\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(m_1)$. In the first case, $m_1 = q\,\hat{}\,w$ for some q and w and hence $m = q\hat{}w\hat{}m_2$. Since m is not in normal form, this is impossible since all rule instances are normal-form ground instances. In the second case, let q and w be the first input and second input of the EXP$\downarrow$ instance that produces $\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(m_1)$. Then $m_1 =_{AC} (q\,\hat{}\,w){\downarrow}_{DH}$ and hence $m =_{AC} ((q\,\hat{}\,w)\,\hat{}\,m_2){\downarrow}_{DH}$. Since w and m_2 are known in *ndg*, $(w * m_2){\downarrow}_{DH}$ is deducible by Lemma 17. Also note that $\mathsf{K}^{\downarrow}_{\mathsf{exp}}(q)$ in *ndg*. Hence, we can apply an instance of either EXP$\uparrow$ or EXP$\downarrow$ of the form

$$\begin{gathered}\mathsf{K}^{d}_{\mathsf{exp}}(q), \mathsf{K}^{\uparrow}_{e}((w * m_2){\downarrow}_{DH}) \\ -[]\!\!\rightarrow \mathsf{K}^{d}_{\mathsf{noexp}}((q\,\hat{}\,(w * m_2)){\downarrow}_{DH})\end{gathered}$$

to deduce m unless it violates **N7**. Then $(q\,\hat{}\,(w * m_2)){\downarrow}_{DH} = a\,\hat{}\,b$ for a public name a and $inp(b) \subseteq inp((w * m_2){\downarrow}_{DH})$. In this case, we can use the construction rule for public names to deduce a. By Lemma 3, $inp((w * m_2){\downarrow}_{DH})$ is deducible since $(w * m_2){\downarrow}_{DH}$ is. Since $inp(b) \subseteq inp((w * m_2){\downarrow}_{DH})$, we can deduce b using PAIR$\uparrow$, INV$\uparrow$, and the multiplication rules. Finally, we can use

$$\mathsf{K}^{\uparrow}_{\mathsf{exp}}(a), \mathsf{K}^{\uparrow}_{e}(b) -[]\!\!\rightarrow \mathsf{K}^{\uparrow}_{\mathsf{noexp}}(a\,\hat{}\,b)$$

to deduce m.

– Nontrivial variant of exp: First, note that for all nontrivial instances of exp, $m_1 = q\,\hat{}\,w$ for some q and w. Assume that m_1 is only in *known*$\uparrow$(*ndg*). Then it must be deduced by the EXP$\uparrow$-rule. Hence, q and w are known and q is not an exponentiation. We can therefore first deduce $(w * m_2){\downarrow}_{DH}$ and then deduce $(q\,\hat{}\,(w * m_2)){\downarrow}_{DH} =_{AC} q\,\hat{}\,(w * m_2){\downarrow}_{DH}$ by applying the EXP$\uparrow$-rule.
If $m_1 \in$ *known*$\downarrow$(*ndg*), but the knowledge fact is tagged with noexp, then we know that m_1 was deduced by an instance of EXP$\downarrow$. Let u and v be the first and second premise of this instance. Then $m_1 =_{AC} (u\,\hat{}\,v){\downarrow}_{DH}$ and we can deduce $(v * m_2){\downarrow}_{DH}$. Hence, we can apply an instance of either EXP$\uparrow$ or EXP$\downarrow$ of the form

$$\begin{gathered}\mathsf{K}^{d}_{\mathsf{exp}}(u), \mathsf{K}^{\uparrow}_{e}((v * m_2){\downarrow}_{DH}) \\ -[]\!\!\rightarrow \mathsf{K}^{d}_{\mathsf{noexp}}((u\,\hat{}\,(v * m_2)){\downarrow}_{DH})\end{gathered}$$

to deduce m unless it violates **N7**. Then $(u\,\hat{}\,(v * m_2)){\downarrow}_{DH} = a\,\hat{}\,b$ for a public name a and $inp(b) \subseteq inp(v * m_2)$. In this case, we can use the construction rule for public names to deduce a. By Lemma 3, $inp((v * m_2){\downarrow}_{DH})$ is deducible since $(v * m_2){\downarrow}_{DH}$ is. Since $inp(b) \subseteq inp((v * m_2){\downarrow}_{DH})$, we can therefore deduce b using PAIR$\uparrow$, INV$\uparrow$, and the multiplication rules. Finally, we can use

$$\mathsf{K}^{\uparrow}_{\mathsf{exp}}(a), \mathsf{K}^{\uparrow}_{e}(b) -[]\!\!\rightarrow \mathsf{K}^{\uparrow}_{\mathsf{noexp}}(a\,\hat{}\,b)$$

to deduce m.

– Variant of mult: Since m_1 and m_2 are known, $m = (m_1 * m_2){\downarrow}_{DH}$ is deducible by Lemma 17.

$\supseteq_{AC}$ **:** We show by induction over dependency graphs that for all $ndg \in ndgraphs(P)$, there is $dg \in dgraphs_{AC}(\lceil P \cup MD\rceil^{DH})$ with dg ${\downarrow}_{DH}$-normal and

$$known(ndg) \subseteq_{AC} known\updownarrow(dg) \quad (1)$$
$$stfacts(ndg) \subseteq^{\sharp}_{AC} stfacts(dg) \quad (2)$$
$$created(ndg) =_{AC} created(dg) \quad (3)$$
$$\overline{trace}(ndg) =_{AC} \overline{trace}(dg). \quad (4)$$

This clearly holds for $([], \varnothing)$. Let $ndg = (I, D) \in ndgraphs(P)$ and $dg = (\tilde{I}, \tilde{D}) \in dgraphs_{AC}(\lceil P \cup MD\rceil^{DH})$ with dg ${\downarrow}_{DH}$-normal such that (1)–(4) hold. Let $ri \in ginsts(\lceil P\rceil^{DH} \cup ND \cup \{\text{FRESH}\})$ arbitrary such that $ndg' = (I \cdot [ri], D \uplus D') \in ndgraphs(P)$. Then we have to show that there is $dg' \in dgraphs_{AC}(\lceil P \cup MD\rceil^{DH})$ with $dg' =_{AC} dg'{\downarrow}_{DH}$ that satisfies (1)–(4) with respect to ndg'.

We perform a case distinction on $l -[\,a\,]\!\!\rightarrow r$.

- *ginsts*(FRESH): Condition (3) for *ndg* and *dg* ensures that we can extend $\tilde{I}$ with ri to obtain dg' without violating unique FRESH instances (**DG4**).
- *ginsts*($\lceil P\rceil^{DH}$): We append ri to $\tilde{I}$ and extend $\tilde{D}$ with the required dependencies to obtain dg' which is possible because of condition (2) for *ndg* and *dg*.
- *ginsts*(*ND*): For every rule in *ND* except for MULT$\uparrow$ and some of the destruction rules for Σ_{ST}, there is a corresponding variant in $\lceil MD\rceil^{DH}$. For a MULT$\uparrow$ instance

$$\begin{gathered}\mathsf{K}^{\uparrow}_{e_1}(t_1), \dots, \mathsf{K}^{\uparrow}_{e_k}(t_k), \mathsf{K}^{\uparrow}_{e_{k+1}}(t_{k+1}), \dots, \mathsf{K}^{\uparrow}_{e_l}(t_l) \\ -[]\!\!\rightarrow \mathsf{K}^{\uparrow}_{\mathsf{mult}}(t_1 * \ldots * t_k * (t_{k+1} * \ldots * t_l)^{-1}),\end{gathered}$$

the messages t_i are known in *dg*. Hence, we can use mult instances

$$\mathsf{K}(t_1 * \ldots * t_j), \mathsf{K}(t_{j+1}) -[]\!\!\rightarrow \mathsf{K}(t_1 * \ldots * t_j * t_{j+1})$$

and inv instances together with mult instances

$$\begin{gathered}\mathsf{K}(t_1 * \ldots * t_k * (t_{k+1} * \ldots * t_{k+j})^{-1}), \mathsf{K}(t^{-1}_{k+j+1}) \\ -[]\!\!\rightarrow \mathsf{K}(t_1 * \ldots * t_k * (t_{k+1} * \ldots * t_{k+j} * t_{k+j+1})^{-1}).\end{gathered}$$

The destruction rules for a function symbol $f \in \Sigma_{ST}$ and a rewriting rule $(f(t_1, \dots, t_k) \to t_i|_p) \in R_{ST}$ can be simulated with the variant $\mathsf{K}(t_1), \dots, \mathsf{K}(t_k) -[]\!\!\rightarrow \mathsf{K}(t_i|p)$ and the construction rules for function symbols that occur in t_i.

■

4) Properties of Normal Dependency Graphs: We use the *extended set of deconstruction rules ND^{destr}* that consists of the deconstruction and exponentiation rules from Figure 9 including the deconstruction rules for Σ_{ST}. We partition the construction rules into the *implicit construction rules $ND^{c\text{-}impl}$* consisting of the pair, inversion, and multiplication construction rules and the *explicit construction rules $ND^{c\text{-}expl}$* consisting of the remaining construction rules (including those for Σ_{ST}) and the COERCE rule.

Lemma (Justification of Lemma 2 on page 9)**.** *For every premise p with fact $K^{\downarrow}_{e}(t)$ of dg, there is a node i in dg such that $I_i \in ginsts(\text{IRECV})$ and $(i,1) \dashrightarrow_{dg} p$.*

Proof: By induction over the length of the rewriting sequences of the normal dependency graphs for P.

The case for $([],\varnothing)$ is trivial.

Assume there is a normal dependency graph $dg = (I,D)$ for P that satisfies the induction hypothesis and there is a rule instance $ri \in ginsts(\lceil P\rceil^{DH} \cup ND \cup \{\text{FRESH}\})$ and a set of dependencies D' disjoint from D such that $dg' = (I \cdot [ri\,], D \uplus D')$ is also a normal dependency graph for P. We show that dg' also satisfies our claim.

The only non-trivial case stems from premises p with fact $K^{\downarrow}_{e}(t)$ of dg' that are not premises of dg. Due to **DG1-2** and the structure of the rules in *ND*, there is a conclusion $(i,1)$ with fact $K^{\downarrow}_{e_1}(m)$ in dg such that $(i,1) \rightarrowtail p \in D'$ and either $I_i \in ginsts(\text{IRECV})$ or $I_i \in ginsts(ND^{destr})$. If $I_i \in ginsts(\text{IRECV})$, then we have $(i,1) \dashrightarrow_{dg'} p$ due to (a) in the definition of $\dashrightarrow_{dg'}$, which concludes this case. If $I_i \in ginsts(ND^{destr})$, then $(i,1)$ is a premise of dg and there is a message t' such that $K^{\downarrow}_{e_2}(t')$). Applying the induction hypothesis yields a node index i' of dg such that $I_{i'} \in ginsts(\text{IRECV})$ and $(i',1) \dashrightarrow_{dg} (i,1)$. Its easy to see that that we can also extend $\dashrightarrow_{dg}$ at the back. Hence, extending $(i',1) \dashrightarrow_{dg} (i,1)$ with $(i,1) \rightarrowtail p$ yields a witness for $(i',1) \dashrightarrow_{dg'} p$, which concludes the proof. ■

Lemma (Justification of Lemma 3 on page 9)**.** *For every premise p in dg with fact $K^{\uparrow}_{e}(t)$ and every message $m \in_{AC} inp(t)$ with $m \neq_{AC} t$, there is a conclusion $(i,1)$ in dg with fact $K^{\uparrow}_{e'}(m')$ such that $I_i \in ginsts(ND^{c\text{-}expl})$, $m' =_{AC} m$, and $(i,1) \twoheadrightarrow_{dg} p$.*

Proof: By induction over the length of the rewriting sequences of the normal dependency graphs for P.

The case for $([],\varnothing)$ is trivial.

Assume there is a normal dependency graph $dg = (I,D)$ for P that satisfies the induction hypothesis and there is a rule instance $ri \in ginsts(\lceil P\rceil^{DH} \cup ND \cup \{\text{FRESH}\})$ and a set of dependencies D' disjoint from D such that $dg' = (I \cdot [ri\,], D \uplus D')$ is also a normal dependency graph for P. We show that dg' also satisfies our claim.

The only non-trivial case stems from premises p with fact $K^{\uparrow}_{e}(t)$ in dg' that are not premises of dg and messages m such that $m \in_{AC} inp(t)$ and $m \neq_{AC} t$. Due to **DG1-2** and the structure of the rules in *ND*, there is a conclusion $(i,1)$ with fact $K^{\uparrow}_{e_1}(m'))$ in dg such that $(i,1) \rightarrowtail p \in D'$ and $m' =_{AC} t$.

As $inp(m') =_{AC} inp(t)$, we have that $m \in_{AC} inp(m')$ and $m \neq_{AC} m'$. Hence, m' is either a pair, an inverse, or a product. In every case, the structure of the rules in *ND* and **N2-4** imply that $I_i \in ginsts(ND^{c\text{-}impl})$. Hence, there are v, e_2, and t' such that (i,v) is a premise with fact $K^{\uparrow}_{e_2}(t')$ in dg. Moreover, $m \in_{AC} inp(t')$ because of the structure of the rules in $ND^{c\text{-}impl}$ and $m \in_{AC} inp(m')$. We make a case distinction on whether $m =_{AC} t'$.

If $m =_{AC} t'$, then there is a conclusion $(j,1)$ with fact $K^{\uparrow}_{e_2}(t')$ modulo AC in dg such that $(j,1) \rightarrowtail (i,v) \in D$. Moreover, $I_j \in ginsts(ND^{constr})$ due to the structure of the rules in *ND*. Hence, $(j,1) \twoheadrightarrow_{dg} p$ due to case (a) of the definition of $\twoheadrightarrow_{dg}$, which concludes this case.

If $m \neq_{AC} t'$, then applying the induction hypothesis to $K^{\uparrow}_{e_2}(t')$ and $m \in_{AC} inp(t')$ yields a conclusion $(i',1)$ with fact $K^{\uparrow}_{e_2}(s))$ in dg such that $s =_{AC} m$, $I_{i'} \in ginsts(ND^{constr})$, and $(i',1) \twoheadrightarrow_{dg} (i,v)$. Extending $(i',1) \twoheadrightarrow_{dg} (i,v)$ with $(i,1) \rightarrowtail p$ yields $(i',1) \twoheadrightarrow_{dg} p$ according to case (b) of the definition of $\twoheadrightarrow_{dg}$, which concludes this proof. ■

E. Proofs for Section VI

In this section, we first give the proof of Theorem 1 from Subsection VI-A. Afterwards, we give the proofs of Theorems 2 and 3 from Section VI.

Before we can prove Theorem 1, we prove some results about the semantics of formulas.

Lemma 19. *For all equational theories E, traces tr, and closed formulas ϕ, $tr \vDash_E \phi$ if and only if $\overline{tr} \vDash_E \phi$.*

Proof: We prove the lemma by induction on the number of silent actions in tr. If there are no silent actions in tr, then $tr = \overline{tr}$ and the statement holds trivially. If tr contains n silent actions and the first silent action occurs at position k, then it suffices to show that $tr \vDash_E \phi$ if and only if $tr' \vDash_E \phi$ for $tr' = tr_1, \dots, tr_{k-1}, tr_{k+1}, \dots, tr_{|tr|}$ since we can then use the induction hypothesis on tr'.

To prove this, we first define the function $\tilde{G}$ that will be used to transform valuations for tr to valuations for tr'.

$$\tilde{G}(u) = \begin{cases} u & \text{if } u \leq k-1 \\ (k-1+u)/2 & \text{if } k-1 < u < k+1 \\ u-1 & \text{if } u \geq k+1 \end{cases}$$

$\tilde{G}$ accounts for the removal of the k-th element of tr as can be seen in the following figure. As can be seen in Figure 18, $\tilde{G}$ is strictly increasing and a bijection.

We define G as the function that is equal to $\tilde{G}$ on $\mathbb{Q}$ and the identity on $\mathcal{M}$. We now prove that for all formulas ϕ and for all valuations θ, $(tr,\theta) \vDash_E \phi$ if and only if $(tr', G \circ \theta) \vDash_E \phi$. This is sufficient since G is a bijection. We use induction over formulas to prove the statement. Let θ be arbitrary, then we have to consider the following cases.

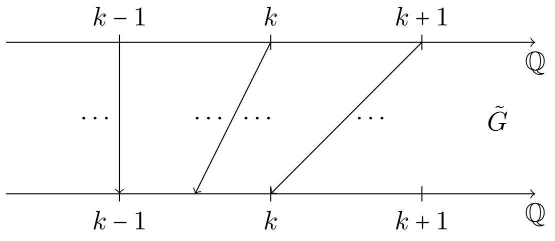


Figure 18. Function graph of $\tilde{G}$.

$f@i$:

$$\begin{array}{lll} & (tr,\theta) \vDash_E f@i & \\ \text{iff} & \theta(i) \in \mathit{idx}(tr) \textit{ and } f\theta \in_E tr_{\theta(i)} & \\ \text{iff} & \begin{array}{l} G(\theta(i)) \in \mathit{idx}(tr') \textit{ and} \\ \quad f(G\circ\theta) \in_E tr'_{G(\theta(i))} \end{array} & [*] \\ \text{iff} & (tr', G\circ\theta) \vDash_E f@i & \end{array}$$

(*) First, note that $f\theta = f(G\circ\theta)$ since G is the identity on $\mathcal{M}$. Then perform a case distinction on $\theta(i)$. If $\theta(i) \notin \mathbb{N}$, then $G(\theta(i)) \notin \mathbb{N}$ and therefore $\theta(i) \notin \mathit{idx}(tr)$ and $G(\theta(i)) \notin \mathit{idx}(tr')$. If $\theta(i) \in \mathbb{N} \setminus \{k\}$, then $G(\theta(i)) \in \mathbb{N}$ and $tr_{\theta(i)} = tr_{G(\theta(i))}$. If $\theta(i) = k$, then $tr_{\theta(i)} = \varnothing$ and $G(\theta(i)) \notin \mathit{idx}(tr')$.

$i \lessdot j$:

$$\begin{array}{lll} & (tr,\theta) \vDash_E i \lessdot j & \\ \text{iff} & \theta(i) < \theta(j) & \\ \text{iff} & G(\theta(i)) < G(\theta(j)) & [G|_{\mathbb{Q}} \textit{ strictly increasing}] \\ \text{iff} & (tr', G\circ\theta) \vDash_E i \lessdot j & \end{array}$$

$i \doteq j$:
Directly follows from the fact that G is a bijection on $\mathbb{Q}$.

$t \approx s$:
Directly follows from the fact that G is the identity on $\mathcal{M}$.

$\exists x.\phi$:

$$\begin{array}{lll} & (tr,\theta) \vDash_E \exists i.\phi & \\ \text{iff} & \begin{array}{l} \textit{there is } u \textit{ such that} \\ (tr, \theta[i \mapsto u]) \vDash_E \phi \end{array} & \\ \text{iff} & \begin{array}{l} \textit{there is } u \textit{ such that} \\ (tr', G\circ(\theta[i\mapsto u])) \vDash_E \phi \end{array} & [\textit{by IH}] \\ \text{iff} & \begin{array}{l} \textit{there is } u' \textit{ such that} \\ (tr', (G\circ\theta)[i\mapsto u']) \vDash_E \phi \end{array} & [u' = G(u)] \\ \text{iff} & (tr', G\circ\theta) \vDash_E \exists i.\phi & \end{array}$$

$\phi \wedge \psi$, $\neg\phi$:
Trivially follow from induction hypothesis.

■

Thanks to the above Lemma, we can consider trace sets TR with $\overline{TR} = \overline{traces}(P)$ for reasoning about the security of a protocol P.

To evaluate guarded trace properties modulo AC, we prove the following theorem which relies on the fact that guarded trace properties only contain variables and public names.

Lemma 20. *Let tr be a trace and ϕ a guarded trace property, then*

$$tr \vDash_{E_{DH}} \phi \quad \textit{iff} \quad tr\!\downarrow_{DH} \vDash_{AC} \phi\ .$$

To prove Lemma 20, we require the following lemma.

Lemma 21. *Let ϕ arbitrary such that for all terms t that occur in ϕ, $t \in \mathcal{V} \cup PN$, tr arbitrary, and θ and ξ valuations such that the following holds for all $x, y \in \mathit{fvars}(\phi)$:*

(1) $\theta(x) \in_{AC} St(tr) \cup PN$ iff $\xi(x) \in_{AC} St(tr) \cup PN$
(2) If $\theta(x) \in_{AC} St(tr) \cup PN$, then $\xi(x) =_{AC} \theta(x)$.
(3) $\theta(x) =_{AC} \theta(y)$ iff $\xi(x) =_{AC} \xi(y)$
(4) $\theta|_{\mathcal{V}_{temp}} = \xi|_{\mathcal{V}_{temp}}$

Then $(tr,\theta) \vDash_{AC} \phi$ if and only if $(tr,\xi) \vDash_{AC} \phi$.

Proof: We prove this by induction on formulas. Let tr arbitrary and θ and ξ such that (1)–(4) hold. We have to consider the following cases.

$F(t_1,\ldots,t_k)@i$ (for $t_i \in \mathcal{V}_{msg} \cup PN$):

$$\begin{array}{lll} & (tr,\theta) \vDash_{AC} F(t_1,\ldots,t_k)@i & \\ \text{iff} & \theta(i) \in \mathit{idx}(tr) \textit{ and } F(t_1,\ldots,t_k)\theta \in_{AC} tr_{\theta(i)} & \\ \text{iff} & \xi(i) \in \mathit{idx}(tr) \textit{ and } F(t_1,\ldots,t_k)\xi \in_{AC} tr_{\xi(i)} & [*] \\ \text{iff} & (tr,\xi) \vDash_{AC} F(t_1,\ldots,t_k)@i & \end{array}$$

(*) First, note that θ and ξ are equal on temporal variables because of (4) and hence $\theta(i) = \xi(i)$. To see that $F(t_1,\ldots,t_k)\theta \in_{AC} tr_{\theta(i)}$ iff $F(t_1,\ldots,t_k)\xi \in_{AC} tr_{\xi(i)}$, we perform a case distinction. If there is a variable x such that $x = t_j$ for some j and $\theta(x) \notin_{AC} St(tr)$, then $\xi(x) \notin_{AC} St(tr)$ because of (1). Hence both facts cannot be elements of the trace. If $\theta(x) \in_{AC} St(tr)$ for all such variables then the same holds for $\xi(x)$ because of (1) and hence θ and ξ agree on these variables because of (2).

$i \lessdot j$, $i \doteq j$:
Holds because of (4).

$x \approx y$:

$$\begin{array}{lll} & (tr,\theta) \vDash_{AC} x \approx y & \\ \text{iff} & \theta(x) =_{AC} \theta(y) & \\ \text{iff} & \xi(x) =_{AC} \xi(y) & [\textit{because of (3)}] \\ \text{iff} & (tr,\xi) \vDash_{AC} x \approx y & \end{array}$$

$n_1 \approx n_2$ (for $n_1, n_2 \in PN$):
Independent of θ and ξ.

$x \approx n$ or $n \approx x$ (for $n \in PN$):
$\theta(x) =_{AC} n$ iff $\xi(x) =_{AC} n$ because of (1) and (2).

$\exists x.\phi$:

$$\begin{array}{ll} & (tr,\theta) \vDash_{AC} \exists x.\phi \\ \text{iff} & \textit{exists } u \textit{ such that } (tr,\theta[x \mapsto u]) \vDash_{AC} \phi \\ \text{iff} & \textit{exists } u' \textit{ such that } (tr,\xi[x \mapsto u']) \vDash_{AC} \phi \quad [*] \\ \text{iff} & (tr,\theta) \vDash_{AC} \exists x.\phi \end{array}$$

(*) We choose u' such that $\theta' = \theta[x \mapsto u]$ and $\xi' = \xi[x \mapsto u']$ satisfy (1)–(4) and we can use the induction hypothesis. If $u \in St(tr) \cup PN$ or $u \in \mathbb{Q}$ we set $u' = u$ and (1)–(4) are obviously satisfied. In all the remaining cases, x is of sort *msg* or *fresh*. If there is $y \in \mathit{fvars}(\phi) \setminus \{x\}$ with $\theta(y) =_{AC} u$, we set $u' = \xi(y)$ to preserve (3). (1), (2), and (4) are also satisfied. If neither is the case, then we can set u' to an arbitrary new message of the right sort that is not in $\xi(\mathit{fvars}(\phi)) \cup St(tr) \cup PN$. Then (1) and (2) hold because they hold for θ and ξ and both u and u' are not in $St(tr) \cup PN$. (3) holds because $\theta'(x) \neq_{AC} \theta'(y)$ for all $y \in \mathit{fvars}(\phi) \setminus \{x\}$ and the same holds for ξ'. (4) holds obviously since $\theta|_{\mathcal{V}_{temp}} = \theta'|_{\mathcal{V}_{temp}}$ and $\xi|_{\mathcal{V}_{temp}} = \xi'|_{\mathcal{V}_{temp}}$.

$\phi \wedge \psi$, $\neg\phi$: Trivially follow from induction hypothesis.

■

Proof of Lemma 20: We show that for all all valuations θ and for all ϕ such that for all terms t that occur in ϕ, $t \in \mathcal{V} \cup PN$,

$$(tr,\theta) \vDash_{E_{DH}} \phi \quad \text{iff} \quad (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}) \vDash_{AC} \phi.$$

We prove this by induction on formulas. Let θ arbitrary, then we have to consider the following cases.

$F(t_1,\dots,t_k)@i$ (for $t_i \in \mathcal{V}_{msg} \cup PN$):

$$\begin{array}{ll} & (tr,\theta) \vDash_{E_{DH}} F(t_1,\dots,t_k)@i \\ \text{iff} & \theta(i) \in idx(tr) \textit{ and} \\ & F(t_1,\dots,t_k)\theta \in_{E_{DH}} tr_{\theta(i)} \\ \text{iff} & \theta(i) \in idx(tr{\downarrow_{DH}}) \textit{ and} \\ & F(t_1,\dots,t_k)(\theta{\downarrow_{DH}}) \in_{AC} (tr{\downarrow_{DH}})_{\theta(i)} \\ \text{iff} & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}) \vDash_{AC} F(t_1,\dots,t_k)@i \end{array}$$

$i \lessdot j$, $i \doteq j$:

Directly follows from $\theta|_{\mathcal{V}_{temp}} = (\theta{\downarrow_{DH}})|_{\mathcal{V}_{temp}}$.

$x \approx y$:

$$\begin{array}{ll} & (tr,\theta) \vDash_{E_{DH}} x \approx y \\ \text{iff} & \theta(x) =_{E_{DH}} \theta(y) \\ \text{iff} & \theta(x){\downarrow_{DH}} =_{AC} \theta(x){\downarrow_{DH}} \\ \text{iff} & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}) \vDash_{AC} x \approx y \end{array}$$

$n_1 \approx n_2$ (for $n_1, n_2 \in PN$):

$n_1 =_{AC} n_2$ iff $n_1 =_{E_{DH}} n_2$.

$x \approx n$ or $n \approx x$ (for $n \in PN$):

$\theta(x) =_{E_{DH}} n$ iff $\theta(x){\downarrow_{DH}} =_{AC} n$.

$\exists x.\phi$: We prove both directions separately.

$$\begin{array}{lll} & (tr,\theta) \vDash_{E_{DH}} \exists x.\phi & \\ \text{implies} & \textit{exists } u \textit{ such that} & \\ & (tr,\theta[x \mapsto u]) \vDash_{E_{DH}} \phi & \\ \text{implies} & \textit{exists } u \textit{ such that} & [IH] \\ & (tr{\downarrow_{DH}}, (\theta[x \mapsto u]){\downarrow_{DH}}) \vDash_{AC} \phi & \\ \text{implies} & \textit{exists } u' \textit{ such that} & [u' = u{\downarrow_{DH}}] \\ & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}[x \mapsto u']) \vDash_{AC} \phi & \\ \text{implies} & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}) \vDash_{AC} \exists x.\phi & \end{array}$$

$$\begin{array}{lll} & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}) \vDash_{AC} \exists x.\phi & \\ \text{implies} & \textit{exists } u \textit{ such that} & \\ & (tr{\downarrow_{DH}}, \theta{\downarrow_{DH}}[x \mapsto u]) \vDash_{AC} \phi & \\ \text{implies} & \textit{exists } u' \textit{ such that} & \\ & (tr, \theta{\downarrow_{DH}}[x \mapsto u']) \vDash_{AC} \phi & \\ & \textit{and } u' \ {\downarrow_{DH}}\text{-normal} & [*] \\ \text{implies} & \textit{exists } u' \textit{ such that} & \\ & (tr,\theta[x \mapsto u]) \vDash_{E_{DH}} \phi & [IH] \\ \text{implies} & (tr,\theta) \vDash_{E_{DH}} \exists x.\phi & \end{array}$$

(*) If u is ${\downarrow_{DH}}$-normal or in $\mathbb{Q}$, then we set $u' = u$. In all the remaining cases, x is of sort *msg*. For these cases, we choose u' such that conditions (1)–(4) from Lemma 21 are satisfied for $\theta{\downarrow_{DH}}[x \mapsto u]$ and $\theta{\downarrow_{DH}}[x \mapsto u']$ and u' ${\downarrow_{DH}}$-normal. We can therefore choose an arbitrary u' such that $u' \notin_{AC} ran(\theta{\downarrow_{DH}}) \cup St(tr) \cup PN$ and u' ${\downarrow_{DH}}$-normal. Since u is not in $St(tr) \cup PN$, (1) and (2) are satisfied. Since $u, u' \notin_{AC} ran(\theta{\downarrow_{DH}})$, (3) is also satisified. Since $x \notin \mathcal{V}_{temp}$, (4) is also satisified.

$\phi \wedge \psi$, $\neg\phi$: Trivially follows from induction hypothesis.

■

Theorem (Justification of Theorem 1)**.** *For every $*$-restricted protocol P and every guarded trace property φ,*

$$P \vDash_{E_{DH}} \varphi \quad \textit{iff} \quad \{trace(dg) \mid dg \in ndgraphs(P)\} \vDash_{AC} \varphi \,.$$

Proof:

$$\begin{array}{lr} P \vDash_{E_{DH}} \varphi & \\ \text{iff } traces(P) \vDash_{E_{DH}} \varphi & \text{[unfold def.]} \\ \text{iff } \overline{traces}(P) \vDash_{E_{DH}} \varphi & \text{[Lemma 19]} \\ \text{iff } \overline{traces}(P){\downarrow_{DH}} \vDash_{AC} \varphi & \text{[Lemma 20]} \\ \text{iff } \{\overline{trace}(dg) \mid dg \in ndgraphs(P)\} \vDash_{AC} \varphi & \text{[Lemma 1]} \\ \text{iff } \{trace(dg) \mid dg \in ndgraphs(P)\} \vDash_{AC} \varphi & \text{[Lemma 19]} \end{array}$$

■

In the completeness proof for the constraint reduction rule **N1**, we require the following lemma to justify that a non-$\downarrow_{DH}$-normal term cannot become a $\downarrow_{DH}$-normal term under substitution.

Lemma 22. *For every valuation θ and every $t \in \mathcal{M}$, if t is not $\downarrow_{DH}$-normal, then $t\theta$ is not $\downarrow_{DH}$-normal.*

Proof: If t is reducible, then there is a position p in t that AC-matches the left-hand side of a rewriting rule in DH. Hence, $t\theta|_p$ is also reducible, as the same rule also AC-matches $t\theta$ at position p. ■

In the completeness proof for the constraint reduction rule **N6**, we require an additional lemma, which intuitively states that every input component of a $\mathsf{K}^{\uparrow}$-premise is a $\mathsf{K}^{\uparrow}$-conclusion of some earlier node. To state this lemma, we define the *extended input components of a term* t as *einp*(t), such that *einp*$(t^{-1}) = \{t^{-1}\} \cup$ *einp*(t), *einp*$(\langle t_1, t_2\rangle) = \{\langle t_1, t_2\rangle\} \cup$ *einp*$(t_1) \cup$ *einp*(t_2), *einp*$(t_1 * t_2) = \{t_1 * t_2\} \cup$ *einp*$(t_1) \cup$ *einp*(t_2), and *einp*$(t) = \{t\}$ otherwise. The difference between *einp*(t) and *inp*(t) is that *einp*(t) additionally retains the deconstructed terms. Hence, *inp*$(t) \subseteq$ *einp*(t) for every term t.

Lemma 23. *For every premise (j, v) in dg with fact $\mathsf{K}^{\uparrow}_{e}(t)$ and every message $m \in_{AC}$ einp(t) there is a conclusion $(i, 1)$ in dg with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ such that $i < j$ and $m' =_{AC} m$.*

Proof: By induction over the length of the rewriting sequences of the normal dependency graphs for P.

The case for $([], \varnothing)$ is trivial.

Assume there is a normal dependency graph $dg = (I, D)$ for P that satisfies the induction hypothesis and there is a rule instance $ri \in$ *ginsts*$(\lceil P\rceil^{DH} \cup ND \cup \{\text{FRESH}\})$ and a set of dependencies D' disjoint from D such that $dg' = (I \cdot [ri], D \uplus D')$ is also a normal dependency graph for P. We show that dg' also satisfies our claim.

The only non-trivial case stems from premises (j, v) with fact $\mathsf{K}^{\uparrow}_{e}(t)$ in dg' that are not premises of dg and messages m such that $m \in_{AC}$ *einp*(t). Due to **DG1-2**, there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\uparrow}_{e_1}(t')$ in dg such that $(i, 1) \rightarrowtail (j, v) \in D'$, $i < j$, and $t' =_{AC} t$. We distinguish whether $m =_{AC} t'$.

If $m =_{AC} t'$, then the proof is complete.

If $m \neq_{AC} t'$, then we have $I_i \in$ *ginsts*(ND^{constr}) due to the structure of the rules in ND, properties **N2-4**, and $m \in_{AC}$ *einp*(t'). Thus, there exists u, e', and m' such that (i, u) is a premise of dg with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ and $m \in$ *einp*(m'). We can thus apply the induction hypothesis to obtain k, e'', and m'' such that $(k, 1)$ is a conclusion in dg with fact $\mathsf{K}^{\uparrow}_{e''}(m'')$ and $k < i$ and $m =_{AC} m''$. As $k < i < j$, this concludes the proof. ■

Theorem (Justification of Theorem 2 on page 13)**.** *The constraint reduction relation $\rightsquigarrow_P$ is sound and complete; i.e., for every $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, the set of P-solutions of Γ is equal to the union of the sets of P-solutions of all Γ_i for $1 \le i \le n$.*

Proof: We first prove the completeness of all rules from Figure 10 and then their soundness.

Note that several constraint reduction rules require freshly renamed sets of rules. Formally, a *renaming of a rule ru fresh with respect to a constraint system* Γ is a well-sorted bijection $\rho : \mathcal{V} \to \mathcal{V}$ such that *vars*$(ru)\rho \cap$ *vars*$(\Gamma) = \varnothing$. Identifying ρ with its homomorphic extension, the freshly renamed rule is then $(ru)\rho$.

Completeness: Let $\mathfrak{M} = (dg, \theta)$ be an arbitrary P-model of some constraint system Γ matching the left-hand-side and satisfying the side-condition of one of the rules from Figure 10. Let $dg = (I, D)$. Let $M = (trace(dg), \theta)$. We perform a case distinction on the rules.

$\mathtt{S}_{\approx}$ From the rule's side-conditions, we have $M \vDash_{AC} t_1 \approx t_2$. Hence, $t_1\theta =_{AC} t_2\theta$. Therefore, there exists $\sigma \in$ *unify*$_{AC}(t_1 \approx t_2)$ and a valuation ξ such that $\theta(x) =_{AC} (\sigma(x))\xi$ for every $x \in$ *dom*(σ) and $\theta(x) = \xi(x)$ for every $x \in$ *vars*$(\Gamma) \smallsetminus$ *dom*(σ). Such a ξ exists because fresh variables in σ are not in *vars*(Γ) and *unify*$_{AC}(t_1 \approx t_2)$ is a complete set of unifiers for t_1 and t_2. The structure (dg, ξ) is a P-model of $\Gamma\sigma$, as $\Vdash$ ensures that the valuation is applied to every variable.

$\mathtt{S}_{\doteq}$ From the rule's side-conditions, we have $M \vDash_{AC} i \doteq j$. Hence, $\theta(i) = \theta(j)$. Thus, $\mathfrak{M}$ is also a P-model of $\Gamma\{i/j\}$, as $\Vdash$ ensures that the valuation is applied to every variable.

$\mathtt{S}_{@}$ From the rule's side-condition, we have $M \vDash_{AC} f@i$. Hence, $\theta(i) \in$ *idx*(I) and there is k with $f\theta =_{AC}$ *acts*$(I_{\theta(i)})_k$. Note that $I \in$ (*ginsts*$(\lceil P\rceil^{DH} \cup ND \cup \{\text{FRESH}\}))^*$. Only rules in $\lceil P\rceil^{DH} \cup \{\text{ISEND}\}$ have a non-empty sequence of actions. Hence, there exists $ru \in \lceil P\rceil^{DH} \cup \{\text{ISEND}\}$ and a grounding substitution σ such that $I_{\theta(i)} = (ru)\sigma$.
We construct a model of the constraint system

$$\Gamma' = \{i : (ru)\rho, f \approx acts((ru)\rho)_k\} \cup \Gamma$$

where ρ is a renaming of ru fresh with respect to Γ. The constraint system Γ' occurs in the right-hand-side of $\mathtt{S}_{@}$. Moreover, (dg, θ') with

$$\theta' = \theta[\rho(x) \mapsto \sigma(x)]_{x \in dom(\sigma)}$$

is a model of Γ as only fresh variables are updated. It is also a P-model of $i : (ru)\rho$ and $f \approx$ *acts*$((ru)\rho)_k$. Thus, it is a P-model of Γ', which concludes this case.

$\mathtt{S}_{\bot}$ Completeness follows trivially from the definition of $\Vdash$ and the definition of $\vDash_{AC}$ for $\bot$.

$\mathtt{S}_{\neg,\approx}$ From the rule's side-condition, we have $M \vDash_{AC} \neg(t \approx t)$. Thus, $t\theta =_{AC} t\theta$ must not hold, which is a contradiction and concludes this case.

$\mathsf{S}_{\neg,\doteq}$ From the rule's side-condition, we have $M \vDash_{AC} \neg(i \doteq i)$. Thus, $\theta(i) = \theta(i)$ must not hold, which is a contradiction and concludes this case.

$\mathsf{S}_{\neg,@}$ From the rule's side-condition, we have $M \vDash_{AC} \neg(f@i)$. Moreover, there is ri such that $(i : ri) \in \Gamma$ and $f \in acts(ri)$. Hence, $\theta(i) \in idx(I)$ and $I_{\theta(i)} =_{AC} ri\,\theta$. Moreover, $f\theta \in acts(I_\theta(i))$. Thus $M \vDash_{AC} f@i$, which is a contradictions and concludes this case.

$\mathsf{S}_{\neg,\lessdot}$ From the rule's side-condition, we have $M \vDash_{AC} \neg(j \lessdot i)$. Hence, $\theta(j) < \theta(i)$ must not hold, which implies that either p $\theta(i) < \theta(j)$ or $\theta(i) = \theta(j)$. Thus, $\mathfrak{M}$ is a model of one of the constraint systems in the rule's right-hand side, which concludes this case.

$\mathsf{S}_{\vee}$ Completeness follows trivially from the definition of $\Vdash$ and the definition of $\vDash_{AC}$ for $\vee$.

$\mathsf{S}_{\wedge}$ Completeness follows trivially from the definition of $\Vdash$ and the definition of $\vDash_{AC}$ for $\wedge$.

$\mathsf{S}_{\exists}$ From the rule's side-condition, we have $M \vDash_{AC} \exists x{:}s.\,\phi$. Hence, there is $w \in \mathbf{D}_s$ such that $(dg, \theta[x \mapsto w]) \Vdash \phi$. It holds that y is fresh and of sort s. Thus, $(dg, \theta[y \mapsto w])$ is a P-model of $\{\phi\{y/x\}\} \cup \Gamma$ which concludes this case.

$\mathsf{S}_{\forall}$ It is easy to see that, for every trace formula χ, $M \vDash_{AC} (\forall\vec{x}.\chi)$ implies $M \vDash_{AC} \chi\sigma$ for every substitution σ with $dom(\sigma) = set(\vec{x})$. Hence, for the substitution σ with $dom(\sigma) = set(\vec{x})$ from the rule's side-condition, we have that $M \vDash_{AC} (f@i)\sigma \Rightarrow \phi\sigma$. Moreover, there is $(j : l\text{-}[\,a\,]\!\!\rightarrow r) \in \Gamma$ such that $j = \sigma(i)$ and $f\sigma \in_{AC} a$. Moreover, $\theta(j) \in idx(I)$ and $I_{\theta(j)} =_{AC} l\theta\text{-}[\,a\theta\,]\!\!\rightarrow r\theta$. Therefore, $M \vDash_{AC} (f@i)\sigma$. Thus, $M \vDash_{AC} \psi\sigma$, which concludes this case.

U_{lbl} From the rule's side-condition, we have $\theta(i) \in idx(I)$, $ri\,\theta =_{AC} I_{\theta(i)} =_{AC} ri\,'\theta$. Hence, $(dg, \theta) \Vdash ri \approx ri\,'$, which concludes this case.

$\mathsf{DG1}_1$ From the rule's side condition, we obtain an i such that $i \lessdot_\Gamma i$. We can prove by induction over the transitive closure in the definition of $\lessdot_\Gamma$ that $i \lessdot_\Gamma i'$ implies $\theta(i) < \theta(i')$. Hence, $i \lessdot_\Gamma i$ is a contradiction, which concludes this case.

$\mathsf{DG1}_2$ From the rule's side-condition, we have $c\theta \rightarrowtail p\theta \in D$, $i : ri \in \Gamma$, $j : ri\,' \in \Gamma$, $u \in idx(concs(ri))$, and $v \in idx(prems(ri\,'))$ such that $c = (i, u)$, $p = (j, v)$, $concs(ri)_u = f$ and $prems(ri\,')_v = f'$. Hence, $\theta(i), \theta(j) \in idx(I)$, $ri\,\theta =_{AC} I_{\theta(i)}$, and $ri\,'\theta =_{AC} I_{\theta(j)}$. Due to **DG1**, $f\theta =_{AC} concs(I_{\theta(i)})_u =_{AC} prems(I_{\theta(j)})_v =_{AC} f'\theta$. Thus $(dg, \theta) \Vdash f \approx f'$, which concludes this case.

$\mathsf{DG2}_1$ From the rule's side-condition, we have $(\theta(i), v) \rightarrowtail p\theta \in D$ and $(\theta(j), u) \rightarrowtail p\theta \in D$. As incoming edges are unique (**DG2**), $(\theta(i), v) = (\theta(j), u)$, which concludes this case.

$\mathsf{DG2}_{2,P}$ From the rule's side-condition, we have $j : ri\,' \in \Gamma$, $v \in idx(prems(ri\,'))$ such that $p = (j, v)$, $prems(ri\,')_v = f$ and f is not a $\mathsf{K}^{\uparrow}$-fact or $\mathsf{K}^{\downarrow}$-fact. Hence, $\theta(j) \in idx(I)$, $ri\,'\theta =_{AC} I_{\theta(j)}$, and $prems(I_{\theta(j)})_v =_{AC} f\theta$. From **DG1-2**, we obtain $k \in idx(I)$ and $u \in idx(concs(I_k))$ such that $(k, u) \rightarrowtail (\theta(j), v) \in D$ and $concs(I_k)_u =_{AC} f\theta$. As f is not a $\mathsf{K}^{\uparrow}$-fact or $\mathsf{K}^{\downarrow}$-fact and $I_k \in ginsts(\lceil P\rceil^{DH} \cup ND \cup \{\textsc{Fresh}\})$, there is $ru \in \lceil P\rceil^{DH} \cup \{\textsc{Isend}, \textsc{Fresh}\}$ and a grounding substitution σ such that $I_k = ru\,\sigma$.
We construct a model of the constraint system

$$\Gamma' = \{i : ru\,\rho,\ (i, u) \rightarrowtail p\} \cup \Gamma$$

where ρ is a fresh renaming of ru with respect to Γ. The constraint system Γ' occurs in the right-hand-side of the constraint reduction rule. The structure

$$(dg,\ \theta[i \mapsto k][\rho(x) \mapsto \sigma(x)]_{x \in dom(\sigma)})$$

is a P-model of Γ as only the valuation of fresh variables is changed. It is also a model of $i : ru\,\rho$ and $(i, u) \rightarrowtail p$. Thus, it is a model of Γ'.

$\mathsf{DG3}$ From the rule's side-condition, we have $c\theta \rightarrowtail (\theta(i), v) \in D$ and $c\theta \rightarrowtail (\theta(j), u) \in D$. Moreover, there is $i : ri \in \Gamma$ such that $u \in idx(concs(ri))$ and ri_u is a linear fact. Hence, $\theta i \in idx(I)$, $I_{\theta(i)} =_{AC} ri\,\theta$, and $(ri\,\theta)_u$ is also a linear fact. Due to **DG3**, linear conclusions have at most one outgoing edge in dg. Hence, $(\theta(i), v) = (\theta(j), u)$ which concludes this case.

$\mathsf{DG4}$ From the rule's left-hand-side, we have $\theta(i), \theta(j) \in idx(I)$, $I_{\theta(i)} =_{AC} (\text{-}[\,]\!\!\rightarrow \mathsf{Fr}(m\theta))$, and $I_{\theta(j)} =_{AC} (\text{-}[\,]\!\!\rightarrow \mathsf{Fr}(m\theta))$. Due to **P3**, we thus have $I_{\theta(i)}, I_{\theta(j)} \in_{AC} ginsts(\textsc{Fresh})$. Due to **DG4**, we have $\theta(i) = \theta(j)$, which concludes this case.

$\mathsf{N1}$ From the rule's side-condition, we have $(i : ri) \in \Gamma$ such that ri is not $\downarrow_{DH}$-normal. Hence, $\theta(i) \in idx(I)$ and $ri\,\theta =_{AC} I_{\theta(i)}$. Due to Lemma 22, $I_{\theta(i)}$ is not $\downarrow_{DH}$-normal, which contradicts property **N1** and thus concludes this case.

$\mathsf{N5,6}$ From the rule's side-condition, we have $i : ri \in \Gamma$, and $i' : ri\,' \in \Gamma$ such that $\mathsf{K}^{d}_{e}(t) =_{AC} concs(ri)_1$, and $\mathsf{K}^{d'}_{e'}(t) =_{AC} concs(ri\,')_1$. Hence, $\theta(i), \theta(i') \in idx(I)$, $ri\,\theta =_{AC} I_{\theta(i)}$, $ri\,'\theta =_{AC} I_{\theta(i')}$, $concs(I_{\theta(i)})_1 =_{AC} \mathsf{K}^{d}_{e}(t\theta)$, and $concs(I_{\theta(i')})_1 =_{AC} \mathsf{K}^{d'}_{e'}(t\theta)$. Moreover, from the rule's side-condition, we also have that either $d = d'$ or $\{i, j\} \cap \{k \mid \exists ri \in insts(\{\textsc{pair}\uparrow, \textsc{inv}\uparrow, \textsc{Coerce}\}).\ (k : ri) \in \Gamma\} = \varnothing$. We make a case distinction whether $d = d'$.
If $d = d'$, then we have $(\theta(i), 1) = (\theta(i'), 1)$ from **N5**, which concludes this subcase.
If $d \neq d'$, then $\{i, j\} \cap \{k \mid \exists ri \in insts(\{\textsc{pair}\uparrow, \textsc{inv}\uparrow, \textsc{Coerce}\}).\ (k : ri) \in \Gamma\} = \varnothing$, which implies $\{ri\,\theta, ri\,'\theta\} \cap ginsts(\{\textsc{Coerce}, \textsc{pair}\uparrow, \textsc{inv}\uparrow\}) = \varnothing$. This contradicts property **N6** of normalized dependency graphs and concludes this subcase and thus the whole case.

N6 From the rule's side-condition, we obtain $i : ri \in \Gamma$, $j : ri' \in \Gamma$, $u \in idx(concs(ri))$, $v \in idx(prems(ri'))$ and $m \in_{AC} inp(t)$ such that $\mathsf{K}^{\downarrow}_{e}(m) =_{AC} concs(ri)_u$ and $\mathsf{K}^{\uparrow}_{e'}(t) =_{AC} prems(ri')_v$. Hence, $\theta(i), \theta(j) \in idx(I)$, $ri\,\theta =_{AC} I_{\theta(i)}$, and $ri'\theta =_{AC} I_{\theta(j)}$. Moreover, $concs(I_{\theta(i)})_u =_{AC} \mathsf{K}^{\downarrow}_{e}(m\theta)$ and $prems(I_{\theta(j)})_v =_{AC} \mathsf{K}^{\uparrow}_{e'}(t\theta)$.
Note that $m \in_{AC} inp(t)$ implies that $m \in_{AC} einp(t)$, which in turn implies that $m\theta \in_{AC} einp(t\theta)$. Thus, we can apply Lemma 23, which yields a conclusion $(k, 1)$ with fact $\mathsf{K}^{\uparrow}_{e_1}(m')$ such that $m' =_{AC} m\theta$ and $k < \theta(j)$. Due to **N6**, we have $\theta(i) < k$. Thus $\theta(i) < \theta(j)$ and $(dg, \theta) \Vdash i \lessdot j$, which concludes this case.

N7 From the rule's side-condition, we have

$$(i : \mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{e}(t_1) \text{-}[]\text{→} \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2 \hat{~} t_2)) \in \Gamma$$

such that s_2 is of sort *pub* and $inp(t_2) \subseteq inp(t_1)$. Thus, $\theta(i) \in idx(I)$ and

$$I_{\theta(i)} =_{AC} \mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1\theta), \mathsf{K}^{\uparrow}_{e}(t_1\theta) \text{-}[]\text{→} \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2\theta \hat{~} t_2\theta) \ .$$

Moreover, $s_2\theta$ is of sort *pub* and $inp(t_2\theta) \subseteq inp(t_1\theta)$, due to $inp(t_2) \subseteq inp(t_1)$. This contradicts **N7**, which concludes this case.

DG2$_{2,\uparrow e}$ Analogous to **DG2**$_{2,P}$ exploiting that $\{m\} = inp(m)$ and m non-trivial imply that m is provided by a construction rule in $ND^{c\text{-}expl}$.

DG2$_{2,\uparrow i}$ From the rule's side-condition, we have $j : ri \in \Gamma$, $v \in idx(prems(ri))$, $s \in \mathcal{T}$, and $m \in inp(s) \smallsetminus (\{s\} \cup PN \cup \cup\{1\} \cup \mathcal{V}_{msg} \cup \mathcal{V}_{pub})$ such that $p = (j, v)$ and $prems(ri)_v = \mathsf{K}^{\uparrow}_{e}(s)$. Hence, $\theta(j) \in idx(I)$ and $ri\,\theta =_{AC} I_{\theta(j)}$. Moreover, $prems(I_\theta(j))_v =_{AC} \mathsf{K}^{\uparrow}_{e}(s\theta)$ and $m\theta \in_{AC} inp(s\theta) \smallsetminus \{s\theta\}$. From Lemma 3, we obtain $k \in idx(I)$, $ru \in ND^{c\text{-}expl}$, a substitution σ grounding for ru, e', and $m \in \mathcal{M}$ such that $I_k = ru\,\sigma$, $concs(ru\,\sigma)_1 = \mathsf{K}^{\uparrow}_{e'}(m')$, $m\theta =_{AC} m'$, and $(k, 1) \twoheadrightarrow_{dg} (\theta(j), v)$. We construct a P-model for the constraint system

$$\Gamma' = \{i : l\rho \text{-}[]\text{→} \mathsf{K}^{\uparrow}_{e'}(t\rho), t\rho \approx m, (i, 1) \twoheadrightarrow p\} \cup \Gamma$$

where $ru = l \text{-}[]\text{→} \mathsf{K}^{\uparrow}_{e'}(t)$ and ρ is a fresh renaming for ru with respect to Γ. The constraint system Γ' occurs in the right-hand-side of the constraint reduction rule. The structure

$$(dg, \theta[i \mapsto k][\rho(x) \mapsto \sigma(x)]_{x \in dom(\sigma)})$$

is a model of $i : l\rho \text{-}[]\text{→} \mathsf{K}^{\uparrow}_{e'}(t\rho)$, $t\rho \approx m$, $(i, 1) \twoheadrightarrow_{dg} p$, and Γ, as only fresh variables are renamed. This concludes this case.

DG2$_{2,\downarrow}$ From the rule's side-condition, we have $j : ri \in \Gamma$, $v \in idx(prems(ri))$ such that $p = (j, v)$ and $prems(ri)_v = \mathsf{K}^{\downarrow}_{e}(m)$. Hence, $\theta(j) \in idx(I)$ and $ri\,\theta =_{AC} I_{\theta(j)}$. Moreover, $prems(I_{\theta(j)})_v =_{AC} \mathsf{K}^{\downarrow}_{e}(m\theta)$. From Lemma 2, we obtain $k \in idx(I)$ such that $I_k = (\text{IRECV})\sigma$ for some grounding grounding substitution σ and $(k, 1) \dashrightarrow_{dg} (\theta(j), v)$. The structure

$$(dg, \theta[i \mapsto k][y \mapsto \sigma(x)]) \ ,$$

where x is the variable x from the IRECV rule, is a model of $i : \mathsf{Out}(y) \text{-}[]\text{→} \mathsf{K}^{\downarrow}_{\mathsf{exp}}(y)$, $(i, 1) \dashrightarrow p$, and Γ, as only fresh variables are changes in the valuation θ. This concludes this case.

DG2$_{\dashrightarrow}$ From the rule's left-hand-side, we obtain $(j, v) = c$ such that $(\theta(j), v) \dashrightarrow_{dg} p\theta$. Due to the definition of $\dashrightarrow_{dg}$, $(\theta(j), v)$ is a $\mathsf{K}^{\downarrow}$-conclusion and either (a) $(\theta(j), v) \rightarrowtail p\theta$ or (b) there is a premise (k, u) such that $(\theta(j), v) \rightarrowtail (k, u)$ and $(k, 1) \dashrightarrow_{dg} p\theta$.
In Case (a), (dg, θ) is a model of the constraint system $\{c \rightarrowtail p\} \cup \Gamma$ in the right-hand-side of the rule.
In Case (b), there is $ru \in ND^{destr}$ and a grounding substitution σ such that $ru\,\sigma = I_k$. Moreover, $u = 1$ and $v = 1$ because of the structure of the rules in ND^{destr}, We construct a model for the constraint system

$$\Gamma' = \{i : ru\,\rho,\ c \rightarrowtail (i, 1),\ (i, 1) \dashrightarrow p,\ \Gamma\}$$

where ρ is a renaming of ru fresh with respect to Γ. The constraint system Γ' occurs in the right-hand-side of the constraint reduction rule. The structure

$$(dg, \theta[i \mapsto k][\rho(x) \mapsto \sigma(x)]_{x \in dom(\sigma)})$$

is a model of Γ', as only fresh variables are changed in the valuation θ, which concludes this case.

Soundness: The only non-trivial cases are the **S**$_\perp$, **S**$_{\doteq}$, **DG2**$_1$, **DG3**, **DG4**, **N5,6**, and **DG2**$_{\dashrightarrow}$ rules, as all other rules only add constraints to the constraint system on the left-hand-side.

S$_\perp$ Soundness follows trivially from the definition of $\Vdash$ and the definition of $\vDash_{AC}$ for $\perp$.

DG2$_{\dashrightarrow}$ We distinguish the following two cases.
Case 1: assume there is a P-model (dg, θ) of

$$\{c \rightarrowtail p\} \cup \Gamma \ .$$

Hence, $c\theta \rightarrowtail_{dg} p\theta$. Moreover, $c\theta \dashrightarrow_{dg} p\theta$. Thus (dg, θ) is a model of $c \dashrightarrow p$, which concludes this case.
Case 2: assume there is a fresh i and $ru \in ND^{destr}$ and a fresh renaming bijection ρ for ru such that there is a P-model $((I, D), \theta)$ of

$$\{i : ru\,\rho,\ c \rightarrowtail (i, 1),\ (i, 1) \dashrightarrow p\} \cup \Gamma \ .$$

Hence, $\theta(i) \in idx(I)$ and $I_{\theta(i)} =_{AC} (ru\,\rho)\theta$. Moreover, $c\theta \rightarrowtail (\theta(i), 1) \in D$ and $(\theta(i), 1) \dashrightarrow_{dg} p\theta$ for $dg = (I, D)$. As $(ru\,\rho)\theta \in_{AC} ginsts(ND^{destr})$, it also holds that $c\theta \dashrightarrow_{dg} p\theta$. Thus (dg, θ) is a model of $c \dashrightarrow p$, which concludes this case.

Soundness for the rules **S**$_{\doteq}$, **S**$_{\neg,\lessdot}$, **DG2**$_1$, **DG3**, **DG4**, and **N5,6** follows from the following argument. Assume that (dg, θ) is a P-model of $\Gamma\{i/j\}$. Then, $(dg, \theta[j \mapsto \theta(i)])$ is a P-model of Γ, which concludes this case.

This concludes the proof that the $\rightsquigarrow_P$-relation is sound and complete.

■

In the following, we prove that, if our algorithm terminates with a solved constraint system, then we can extract an attack on $P \vDash_{E_{DH}} \varphi$. Intuitively, the extraction uses two steps. We first instantiate variables in $\mathcal{V}_{pub}$ with distinct public constants, variables in $\mathcal{V}_{msg} \cup \mathcal{V}_{fresh}$ with distinct fresh constants, and temporal variables with their index according to a topological sorting of $\lessdot_\Gamma$. Then, we extend the $\twoheadrightarrow$-constraints and solve the remaining open premises.

The formal proof requires two auxiliary lemmas and proceeds as follows. We first state six conditions that are required for a successful solution extraction and show that these conditions are invariants of our constraint-reduction relation $\rightsquigarrow_P$. Then, we show how to extract a P-solution from a solved constraint system satisfying these conditions. Finally, we prove the desired theorem that we can extract a P-solution from every solved constraint system Γ returned by our constraint-solving algorithm.

Lemma 24. *The conjunction of the following properties is invariant under the constraint reduction relation $\rightsquigarrow_P$.*

CS1 *Every node is labeled with an instance of a rule in $\lceil P \rceil^{DH} \cup (ND \smallsetminus ND^{c\text{-}impl}) \cup \{\text{FRESH}\}$.*

CS2 *For every edge $c \rightarrowtail p \in \Gamma$ and every chain $c \dashrightarrow p \in \Gamma$, $c \in dom(cs(\Gamma))$ and $p \in dom(ps(\Gamma))$.*

CS3 *There is no $K_e^\uparrow(t)$ conclusion where t is unifiable with a pair, an inversion, or a product.*

CS4 *Every conclusion c with a $K_e^\downarrow(t)$ fact has an outgoing edge or an outgoing chain.*

CS5 *If there is a $K_e^\downarrow(t)$ conclusion and t is a product, then the constraint system is not solved.*

CS6 *All trace formulas $\phi \in \Gamma$ are guarded trace formulas.*

Proof: We justify each property individually.

CS1 All reduction rules adding $i : ri$ constraints ensure that ri is an instance of a rule in $\lceil P \rceil^{DH} \cup (ND \smallsetminus ND^{c\text{-}impl}) \cup \{\text{FRESH}\}$. Moreover, this property is invariant under instantiation, which concludes this case.

CS2 All reduction rules adding $c \rightarrowtail p$ or $c \dashrightarrow p$ constraints ensure this property. Moreover, this property is invariant under instantiation, which concludes this case.

CS3 The only reduction rules adding rule instances with $K_e^\uparrow(t)$ conclusions are $\texttt{DG2}_{2,\uparrow e}$ and $\texttt{DG2}_{2,\uparrow i}$. Their side conditions ensures that t is not unifiable with pairs, inversions, and products. Moreover, this property is invariant under instantiation, which concludes this case

CS4 The only rules adding $K^\downarrow$-conclusion are $\texttt{DG2}_{2,\downarrow}$ and $\texttt{DG2}_{\dashrightarrow}$. Both of them ensure this property. Moreover, this property is invariant under instantiation, which concludes this case.

CS5 Due to **CS4**, every $K_e^\downarrow(t)$ conclusion has either (1) an outgoing edge to some premise with fact f or (2) an outgoing destruction chain.
In case (1), we can either apply the reduction rule $\texttt{DG1}_2$ or $f =_{AC} K_e^\downarrow(t)$. In the second case, t cannot be a product, as all nodes are labeled with instances of rules in $\lceil P \rceil^{DH} \cup (ND \smallsetminus ND^{c\text{-}impl}) \cup \{\text{FRESH}\}$ due to **CS1**.
In case (2), we have $t \notin \mathcal{V}_{msg}$ and we can therefore apply rule $\texttt{DG2}_{\dashrightarrow}$, which implies that the constraint system is not solved.

CS6 If all trace formulas are guarded, then all rules in $\rightsquigarrow_P$ only add guarded trace formulas.

■

Lemma 25. *We can construct a P-model for every solved constraint system Γ satisfying **CS1-6**.*

Proof: We construct a P-model of Γ using the following two steps. First, we construct almost a dependency graph dg that satisfies all properties of a normal dependency graph for P except **DG2**. We also construct a valuation θ such that the structure (dg, θ) satisfies all constraints in Γ except the $\twoheadrightarrow$-constraints. Then, we extend dg such that **DG2** and the $\twoheadrightarrow$-constraints are also satisfied. We construct this extension such that the properties and constraints already satisfied by dg are also satisfied by its extension.

For the first step, we choose some sequence $\vec{j} = [j_1, \ldots, j_n]$ of all temporal variables in Γ ordered according to a topological-sorting of $\lessdot_\Gamma$. Such a sequence exists because rule $\texttt{DG1}_1$ is not applicable. Let $[i_1, \ldots, i_l]$ for $l \le n$ be the subsequence of $\vec{j}$ that consists of all temporal variables i_k with an associated node constraint $i_k : ri_k$ in Γ. Note that the rule instances ri_k are unique modulo AC, as rule $\texttt{U}_{lbl}$ is not applicable. We choose θ to be an injective valuation such that θ maps variables in $\mathcal{V}_{msg}$ to FN[1], the range of θ is disjoint from $St(\Gamma)$, $\theta(i_k) = k$ for $1 \le k \le l$, and $\theta(j_k) < \theta(j_{k+1})$ for $1 \le k < n$. Such a θ exists because there are infinitely many public and fresh names and $\mathbb{Q}$ is dense. We define

$$dg = (D, I) = ([ri_1\theta, \ldots, ri_n\theta], \{(c\theta, p\theta) \mid c \rightarrowtail p \in \Gamma\}) \ .$$

This construction makes (dg, θ) almost a P-model for Γ, as it satisfies the following properties.

It holds that $D \subseteq (\mathbb{N}\times\mathbb{N})\times(\mathbb{N}\times\mathbb{N})$ and $I \in (ginsts(\lceil P \rceil^{DH} \cup ND \cup \{\text{FRESH}\}))^*$. The first proposition holds because **CS2** ensures that every edge uses only temporal variables that are labeled with a rule instance. For the second proposition, we must show that every $ri \in I$ is a ground instance of a rule in $ri \in \lceil P \rceil^{DH} \cup ND \cup \{\text{FRESH}\}$. By construction, there is $(i : ri') \in \Gamma$ such that $ri = ri'\theta$. Due to **CS1**, ri', and hence ri, is an instance of $\lceil P \rceil^{DH} \cup ND \cup \{\text{FRESH}\}$. As the range of θ is ground, we have that $ri = ri'\theta$ is ground.

[1]This means that we instantiate message variables with intruder-generated fresh names.

Moreover, dg satisfies all properties of a normal dependency graph except **DG2**.

DG1 holds. Let $(i,u) \rightarrowtail (j,v) \in D$. By construction, $i < j$ holds. Moreover due to our construction and **CS2**, there are facts f and f' such that $((i,u), f) \in \mathit{concs}(dg)$ and $((j,v), f') \in \mathit{prems}(dg)$. It holds that $f =_{AC} f'$, as rules $\texttt{DG1}_2$ and $\texttt{S}_{\approx}$ are not applicable.

DG3 holds, as rule **DG3** is not applicable.

DG4 holds, as rule **DG4** is not applicable.

N1 We must show that every $ri \in I$ is $\downarrow_{DH}$-normal. By construction, there is $(i : ri') \in \Gamma$ such that $ri = ri'\theta$. Due to rule **N1** not being applicable, ri' is $\downarrow_{DH}$-normal. Hence, $ri'\theta$ is $\downarrow_{DH}$-normal, as all terms in the range of θ are names.

N2 holds because of the structure of COERCE and **CS3**.

N3 holds because of **CS1**, i.e., because there is no instance of the multiplication rule in dg.

N4 holds because of **CS3,5**, i.e., because there are no conclusion facts deriving a product.

N5 holds because rule **N5,6** is not applicable.

N6 holds because rules **N5,6** and **N6** are not applicable.

N7 holds because θ maps all message variables to fresh names and the rule **N7** is not applicable.

It holds that (dg, θ) is a model of all graph constraints in Γ except the $c \twoheadrightarrow_{dg} p$ constraints.

- $(dg, \theta) \Vdash i : ri$ holds for every $i : ri \in \Gamma$ by construction.
- $(dg, \theta) \Vdash c \rightarrowtail p$ holds for every $c \rightarrowtail p \in \Gamma$ by construction.
- $(dg, \theta) \Vdash c \dashrightarrow p$ holds for every $c \dashrightarrow p \in \Gamma$ because there is no chain constraint remaining in a solved constraint system. This holds, as the properties of protocol rules guarantee that for every message variable x in the conclusion of a rule there is also an earlier $\mathsf{K}^{\uparrow}$-premise requiring the message variable x. Hence, if a chain constraint remained, then the rule **N6** were applicable.

It holds that $M = (\mathit{trace}(dg), \theta)$ is a model of all trace formulas in Γ. Using **CS5**, we prove this by induction over the size of the guarded trace formulas in Γ.

- $M \vDash_{AC} t_1 \approx t_2$. As $\texttt{S}_{\approx}$ is not applicable, $t_1 =_{AC} t_2$. Thus, $t_1\theta =_{AC} t_2\theta$, which concludes this case.
- $M \vDash_{AC} i \doteq j$. As $\texttt{S}_{\doteq}$ is not applicable, $i = j$. Thus, $\theta(i) =_{AC} \theta(j)$, which concludes this case.
- $M \vDash_{AC} i \lessdot j$ holds by construction.
- $M \vDash_{AC} f@i$. As $\texttt{S}_{@}$ is not applicable, $f@i \in_{AC} as(\Gamma)$. Hence, there is $i : l\text{-}[\, a \,]\!\rightarrow r \in \Gamma$ such that $f \in \mathit{set}(a)$. By construction, $\theta(i) \in \mathit{idx}(\mathit{trace}(dg))$ and $\mathit{trace}(dg)_{\theta(i)} = \mathit{set}(a\theta)$. Thus, $f\theta \in_{AC} \mathit{trace}(dg)_{\theta(i)}$, which concludes this case.
- $M \vDash_{AC} \neg(t_1 \approx t_2)$. There is no such constraint in Γ, as otherwise rule $\texttt{S}_{\neg,\approx}$ would be applicable.
- $M \vDash_{AC} \neg(i \doteq j)$. There is no such constraint in Γ, as otherwise rule $\texttt{S}_{\neg,\doteq}$ would be applicable.
- $M \vDash_{AC} \neg(i \lessdot j)$. As $\texttt{S}_{\neg,\lessdot}$ is not applicable, we have $j \lessdot_{\Gamma} i$ or $j = i$. In both cases, $M \vDash_{AC} \neg(i \lessdot j)$ holds by construction.
- $M \vDash_{AC} \neg(f@i)$. There is no such constraint in Γ, as otherwise rule $\texttt{S}_{\neg,@}$ would be applicable.
- $M \vDash_{AC} \phi_1 \wedge \phi_2$. Rule $\texttt{S}_{\wedge}$ is not applicable. Hence $\phi_1, \phi_2 \in \Gamma$. As ϕ_1 and ϕ_2 are smaller than $\phi_1 \wedge \phi_2$, the induction hypothesis applies.
- $M \vDash_{AC} \phi_1 \vee \phi_2$. Rule $\texttt{S}_{\vee}$ is not applicable. Hence $\phi_1 \in \Gamma$ or $\phi_2 \in \Gamma$. As ϕ_1 and ϕ_2 are smaller than $\phi_1 \vee \phi_2$, the induction hypothesis applies in either case.
- $M \vDash_{AC} \exists x{:}s.\, \phi$. Rule $\texttt{S}_{\exists}$ is not applicable. Hence there exists a $w{:}s$ such that $\phi\{w/x\} \in \Gamma$. Hence, we have $M \vDash_{AC} \phi\{w/x\}$ from the induction hypothesis. Thus, $(\mathit{trace}(dg), \theta[x \mapsto w\theta]) \vDash_{AC} \phi$, which concludes this case.
- $M \vDash_{AC} \forall \vec{x}.\, \neg(f@i) \vee \psi$. It suffices to show that, for every valuation θ' with $\theta'(x) = \theta(x)$ for all $x \notin \vec{x}$, it holds that $(\mathit{trace}(dg), \theta') \vDash_{AC} f@i$ implies $(\mathit{trace}(dg), \theta') \vDash_{AC} \psi$.
  Let θ' be such a valuation. We will construct a substitution α from θ' such that $\mathit{dom}(\alpha) = \mathit{set}(\vec{x})$ and $(f@i)\alpha \in as(\Gamma)$. Hence, $M \vDash_{AC} \psi\alpha$ because the reduction rule $\texttt{S}_{\forall}$ is not applicable. We will show that our construction of α ensures that $M \vDash_{AC} \psi\alpha$ implies $(\mathit{trace}(dg), \theta') \vDash_{AC} \psi$. This will conclude this case.
  We require an auxiliary definition and two auxiliary claims to prove this case. We define $\hat{\theta} : \mathcal{T} \to \mathcal{T}$ as follows.

$$\hat{\theta}(t) = \begin{cases} \theta^{-1}(t) & \text{if } t \in \mathit{ran}(\theta) \\ t & \text{if } t \in (PN \cup FN \cup \mathcal{V}) \setminus \mathit{ran}(\theta) \\ f(\hat{\theta}(t_1), \ldots, \hat{\theta}(t_k)) & \text{if } t = f(t_1, \ldots, t_k) \text{ for } f \in \Sigma^k \end{cases}$$

Intuitively, $\hat{\theta}$ lifts θ^{-1} to terms and temporal variables.[2]

Claim 1. *For all $t, s \in \mathcal{T}$ with t ground and $St(s) \cap \mathit{ran}(\theta) = \varnothing$, it holds that $t =_{AC} s\theta$ implies $\hat{\theta}(t) =_{AC} s$.*

Proof: By induction over t.

- Case $t \in PN \cup FN$. Case distinction on s.
  - $s \in PN \cup FN$. From $t =_{AC} s\theta$, we have $t = s$. Note that $s \notin \mathit{ran}(\theta)$ by assumption. Thus, $\hat{\theta}(s) =_{AC} s$.
  - $s \in \mathcal{V}$. From $t =_{AC} \theta(s)$, we have $t \in \mathit{ran}(\theta)$. Thus, $\hat{\theta}(t) = \theta^{-1}(t) = s$ because θ is injective.
- $t = f(t_1, \ldots, t_k)$ for $f \in \Sigma^k$ and $t_i \in \mathcal{T}$. Trivial due to $\mathit{ran}(\theta) \subseteq PN \cup FN$, the distributivity of $=_{AC}$ and St over f, and our induction hypothesis.

□

We define $\alpha = \{\hat{\theta}(\theta'(x))/x\}_{x \in \vec{x}}$. Note that α is a (well-sorted) substitution due to the definition of valuations

[2]Here, we assume that terms can also be built from temporal variables. This saves us duplicated proof work.

and the definition of guarded trace formulas that ensures that $\vec{x} \subseteq (\mathcal{V}_{msg} \cup \mathcal{V}_{temp})^*$.

Claim 2. *For all $t, s \in \mathcal{T}$ with $(St(t) \cup St(s)) \cap ran(\theta) = \emptyset$, it holds that $t\theta' =_{AC} s\theta$ implies $t\alpha =_{AC} s$.*

Proof: By induction over t.

- Case $t \in PN \cup FN$. We must show that $t =_{AC} s\theta$ implies $t =_{AC} s$. Case distinction on s.
  * $s \in PN \cup FN$. Trivial.
  * $s \in \mathcal{V}$. We have $t = \theta(s)$. Hence, $t \in ran(\theta)$, which contradicts our assumptions.
- Case $t \in \mathcal{V}$. We must show that $\theta'(t) =_{AC} s\theta$ implies $\alpha(t) =_{AC} s$. We make a case distinction.
  * $t \in \vec{x}$: We have $\theta'(t)$ ground and $St(s) \cap ran(\theta) = \emptyset$. Moreover, $\theta'(t) =_{AC} s\theta$. Hence, $\hat{\theta}(\theta'(t)) =_{AC} s$ holds due to Claim 1. Thus, $\alpha(t) =_{AC} s$, which concludes this case.
  * $t \notin \vec{x}$: We have $\theta(t) =_{AC} s\theta$ due to the assumptions on θ'. We must show $t =_{AC} s$. We make a case distinction on s.
    · $s \in PN \cup FN$. We have $\theta(t) =_{AC} s$. Hence, $s \in ran(\theta)$, which contradicts our assumptions.
    · $s \in \mathcal{V}$. We have $\theta(t) =_{AC} \theta(s)$. Due to the injectivity of θ, we have $t = s$, which concludes this case.
- $t = f(t_1, \dots, t_k)$ for $f \in \Sigma^k$ and $t_i \in \mathcal{T}$. Trivial due to $ran(\theta) \subseteq PN \cup FN$, the distributivity of $=_{AC}$ and St over f, and our induction hypothesis.

□

Note that $(trace(dg), \theta') \vDash_{AC} f@i$. Due to the construction of dg, we hence obtain $j : l \text{-}[\, a \,]\!\rightarrow r \in \Gamma$ and $f' \in a$ such that $(i, f)\theta' =_{AC} (j, f')\theta$. Note that all subterms of f and f' are subterms of Γ. Moreover, $ran(\theta) \subseteq (PN \cup FN) \setminus St(\Gamma)$, which implies that $(St(i, f) \cup St(j, f')) \cap ran(\theta) = \emptyset$. We thus have $(i, f)\alpha =_{AC} (j, f')$ due to Claim 2. Note that $f'@j \in as(\Gamma)$ and hence $(f@i)\alpha \in_{AC} as(\Gamma)$. Thus, $(trace(dg), \theta) \vDash_{AC} \psi\alpha$, as the reduction rule $\mathbf{S}_{\neg,@}$ is not applicable. Hence, $(trace(dg), \theta[x \mapsto (\alpha(x))\theta]_{x \in \vec{x}}) \vDash_{AC} \psi$. Note that $(\alpha(x))\theta = (\hat{\theta}(\theta'(x)))\theta = \theta'(x)$ due to the definition of $\hat{\theta}$. Thus, $(trace(dg), \theta') \vDash_{AC} \psi$, which concludes this case.

Note that the only violations of **DG2** stem from $\mathsf{K}^{\uparrow}$-premises that require either a pair, an inversion, a product, a trivial message, or an intruder generated fresh name in $ran(\theta)$. We can remove these violations by extending (dg, θ) with the missing instances of the corresponding normal message deduction rules and the missing edges between them and other rule instances. This is possible, as the only open premises are open message premises requiring a trivial message or an intruder-generated fresh name. Moreover, the semantics of $c \twoheadrightarrow p$ constraints ensures that there is at least one implicit construction rule in between the conclusion c and the premise p. Therefore, there cannot be a violation of the exponentiation tags when introducing edges for the implicit constructions. We construct this extension by exploiting Lemma 19 to insert further rule instances into I without violating the trace formulas in Γ. We perform this extension such that the uniqueness of K-conclusions is not violated. Thus, we can construct a P-model for every solved constraint system Γ satisfying **CS1-6**. ■

Theorem (Justification of Theorem 3 on page 13)**.** *We can construct a P-solution from every solved system in the state Ω of our constraint-solving algorithm.*

Proof: The properties **CS1-6** are satisfied for the starting state used by our algorithm. They are maintained by our algorithm due to Lemma 24. Hence, we can construct a P-solution for the solved constraint system Γ using Lemma 25. ■