Efficient Decision Procedures for Message Deducibility and Static Equivalence

Bruno Conchinha¹, David Basin², and Carlos Caleiro³

¹ Information Security Group, ETH Zürich, Zürich, Switzerland
brunoco@inf.ethz.ch¹, basin@inf.ethz.ch²
² SQIG - Instituto de Telecomunicações, Department of Mathematics, IST, TU Lisbon, Portugal
ccal@math.ist.utl.pt³

Abstract. We consider two standard notions in formal security protocol analysis: message deducibility and static equivalence under equational theories. We present new polynomial-time algorithms for deciding both notions under subterm convergent equational theories and under a theory representing symmetric encryption with the prefix property. For these equational theories, polynomial-time algorithms for the decision problems associated to both notions are well-known (although this has not been proven for static equivalence under the prefix theory). However, our algorithms have a significantly better asymptotic complexity than existing approaches.

As an application, we use our algorithm for static equivalence to discover off-line guessing attacks on the Kerberos protocol when implemented using a symmetric encryption scheme for which the prefix property holds.

Key words: security protocols, equational theories, deducibility, static equivalence

1 Introduction

Formal methods and associated tools are now well established for analyzing security protocols. In symbolic approaches, the messages exchanged by agents are represented by terms in an algebra and properties of cryptographic operators are formalized equationally. This sets the scene for different analysis approaches.

Message deducibility and static equivalence are two notions of knowledge in the equational setting, both with direct applications to security protocol analysis. Procedures for analyzing the security of cryptographic protocols use algorithms for deduction [5, 6, 14, 17], and static equivalence has been used to study cryptographic indistinguishability [9] and to model and reason about off-line guessing attacks [1, 7, 19].

Contributions In this paper, we give new algorithms for deciding both deduction and static equivalence in two practically relevant cases: subterm convergent theories and theories for symmetric encryption with the prefix property. Our algorithms improve the runtime complexity bounds of existing approaches.
Our algorithms for the prefix theory are a simple generalization of those for subterm convergent theories. To the best of our knowledge, no polynomial-time algorithm for static equivalence under such theories was previously known.

As an application, we analyze the security of the Kerberos protocol against off-line guessing attacks. Although it was previously known that Kerberos admits an off-line guessing attack [23], we are able to find numerous new attacks by considering an implementation with a symmetric encryption scheme for which the prefix property holds, e.g., if ECB or CBC modes of operation are used. Identifying such attacks highlights design and implementation issues (such as message ordering, modes, etc.) that may not appear important. Automating such analysis requires static equivalence procedures for different theories and highlights the importance of efficient, automated methods for this task.

Background and Related Work The notions of deduction and static equivalence that we consider were originally introduced in the context of the pi-calculus [4]. Considerable work has been devoted to proving the decidability of both deduction and static equivalence for a wide range of equational theories. [2] gives polynomial-time algorithms for solving both problems under equational theories generated by subterm convergent rewriting systems. For other equational theories, there are few polynomial-time results, but even then decidability results exist under fairly general conditions, e.g., [3,14–18,21,22,24,25]. Despite the considerable prior work in this area, there have been few implementations of these algorithms, particularly for static equivalence. For some time, ProVerif [10, 11] was the only tool capable of deciding static equivalence. General algorithms for deduction and static equivalence have more recently been implemented by the YAPA [8] and KISS [16] tools. Although the precise set of equational theories and conditions under which these algorithms terminate is unclear, they seem able to handle most theories previously studied. We provide a detailed comparison of our algorithms with those implemented by these tools.

Even less attention has been devoted to improving the efficiency of algorithms for these problems. Given the interest in tools for solving these problems, complexity issues are not only theoretically interesting but also practically relevant.

Organization In Section 2, we introduce basic definitions and notation. In Section 3, we present our algorithms and their properties. In Section 4, we extend the methods of Section 3 to handle symmetric encryption with the prefix property. As a case study, we analyze the security of the Kerberos protocol against off-line guessing attacks. We draw conclusions in Section 5. For reasons of space and readability, full proofs are given in the Appendix.

2 Background and Basic Definitions

Given a function \( g \), we denote by \( \text{dom}(g) \) and by \( \text{ran}(g) \) its domain and range, respectively. When \( X \subseteq \text{dom}(g) \), we write \( g[X] \) for the image of \( X \) under \( g \).

We consider signatures \( \Sigma = \biguplus_{i \in \mathbb{N}} \Sigma_i \) consisting of a finite number of function symbols, where \( \Sigma_i \) contains the functions symbols of arity \( i \). For each \( f \in \Sigma \), the
function \( \alpha : \Sigma \rightarrow \mathbb{N} \) returns the arity \( \alpha(f) \) of \( f \). We also fix infinite, disjoint sets \( \text{Var} \) and \( \text{Name} \) of variables and names. Intuitively, names are used to represent fresh data and constant symbols (i.e., symbols in \( \Sigma_0 \)) represent publicly known constants. We assume that \( x, y, z \in \text{Var} \) and that \( \{ x_i \mid i \in \mathbb{N} \} \subseteq \text{Var} \).

**Example 1.** The signature \( \Sigma^{DY} \), representing a Dolev-Yao model with a hash function \( h \), a pairing function \( \text{pair} \), the projections \( \pi_1 \) and \( \pi_2 \), and symmetric and asymmetric encryption and decryption, is given by \( \Sigma^{DY} = \Sigma_1^{DY} \cup \Sigma_2^{DY} \), where 

\[
\Sigma_1^{DY} = \{ h, \pi_1, \pi_2, \text{pub}, \text{priv} \} \quad \text{and} \quad \Sigma_2^{DY} = \{ \text{enc}_{\text{sym}}, \text{dec}_{\text{sym}}, \text{pair}, \text{enc}_{\text{asym}}, \text{dec}_{\text{asym}} \}.
\]

\( \text{pub} \) and \( \text{priv} \) represent the generation of public and private keys. We use the following abbreviations: \( \langle x, y \rangle \) for \( \text{pair}(x, y) \); \( \{ P \}_{K} \) for \( \text{enc}_{\text{sym}}(P, K) \); \( \{ C \}_{K}^{-1} \) for \( \text{dec}_{\text{sym}}(C, K) \); \( x_{\text{pub}} \) for \( \text{pub}(x) \); \( x_{\text{priv}} \) for \( \text{priv}(x) \); \( \{ P \}_{K} \) for \( \text{enc}_{\text{asym}}(P, K) \); \( \{ C \}_{K}^{-1} \) for \( \text{dec}_{\text{asym}}(C, K) \); and \( \langle x_1, \ldots, x_n \rangle \) for \( \langle \ldots \langle x_1, x_2 \rangle \ldots, x_n \rangle \).

As usual, given a set \( X \), \( T(\Sigma, X) \) is the set of \( \Sigma \)-terms over \( X \), i.e., the smallest set such that \( X \subseteq T(\Sigma, X) \) and \( f(t_1, \ldots, t_n) \in T(\Sigma, X) \) for all \( t_1, \ldots, t_n \in T(\Sigma, X) \) and all \( f \in \Sigma_n \). We use the symbol \( = \) to denote syntactic equality. Given \( t \in T(\Sigma, X) \), we define the set \( \text{sub}(t) \) of subterms of \( t \) as usual: if \( t \in X \), then \( \text{sub}(t) = \{ t \} \); if \( t = f(t_1, \ldots, t_n) \) for some \( f \in \Sigma_n \) and \( t_1, \ldots, t_n \in T(\Sigma, X) \), then \( \text{sub}(t) = \{ t \} \cup \bigcup_{i=1}^{n} \text{sub}(t_i) \). We denote by \( \text{vars}(t) = \text{sub}(t) \cap \text{Var} \) the set of variables occurring in \( t \).

We use the standard notion of substitution as a partial function \( \sigma : \text{Var} \rightarrow T(\Sigma, X) \). We abuse notation by using the same symbol \( \sigma \) for a substitution and its homomorphic extension to \( T(\Sigma, X) \), where \( \text{dom}(\sigma) \subseteq X \). As usual, we write \( t\sigma \) instead of \( \sigma(t) \).

A frame is a pair \( (\hat{n}, \sigma) \), written \( \nu \hat{n} \sigma \), where \( \hat{n} \subseteq \text{Name} \) is a finite set of names and \( \sigma : \text{Var} \rightarrow T(\Sigma, \text{Name}) \) is a substitution with finite domain. Intuitively, names in \( \hat{n} \) represent fresh data generated by other agents and thus unavailable to the attacker, while \( \sigma \) represents the messages learned by the attacker by eavesdropping on the network. Given a frame \( \phi = \nu \hat{n} \sigma \), we define \( T_\phi = T(\Sigma, (\text{Name} \setminus \hat{n}) \cup \text{dom}(\sigma)) \). We say that terms in \( T_\phi \) are \( \phi \)-recipes. The terms in \( \sigma[T_\phi] \) are the concrete terms that the attacker can obtain and we refer to them as terms constructible from \( \phi \).

A rewriting system \( R \) over \( \Sigma \) is a set of rewrite rules of the form \( l \rightarrow r \), where \( l, r \in T(\Sigma, \text{Var}) \). We assume that rewriting systems have only finitely many rules. Given a rewriting system \( R \), we define the relation \( \rightarrow_R \subseteq T(\Sigma, \text{Name}) \times T(\Sigma, \text{Name}) \) as the smallest relation such that:

- if \( (l \rightarrow r) \in R \) and \( \sigma : \text{vars}(l) \rightarrow T(\Sigma, \text{Name}) \) is a substitution, then \( l\sigma \rightarrow_R r\sigma \), and
- if \( t_1, \ldots, t_n, t'_i \in T(\Sigma, \text{Name}) \), \( t_i \rightarrow_R t'_i \), and \( f \in \Sigma_n \), then \( f(t_1, \ldots, t, \ldots, t_n) \rightarrow_R f(t_1, \ldots, t'_i, \ldots, t_n) \).

We drop the \( R \) from \( \rightarrow_R \) when it is clear from context.

A rewriting system \( R \) is convergent if it is terminating and confluent. In this case, each term \( f \) has a unique normal form \( t_{\downarrow R} \in T(\Sigma, \text{Name}) \). Given a convergent rewriting system \( R \), we define \( \approx_R \subseteq T(\Sigma, \text{Name}) \times T(\Sigma, \text{Name}) \) as
the relation such that \( t \approx_R t' \) if and only if \( t \downarrow_R = t' \downarrow_R \). Note that we adopt the usual convention of writing \( t \approx_R t' \) instead of \( (t, t') \in \approx_R \). It is simple to check that \( \approx_R \) is an equational theory (i.e., an equivalence relation closed under the application of contexts). We call \( \approx_R \) the *equational theory generated by* \( R \). A rewriting system \( R \) is subterm convergent if it is convergent and, for each \((l \to r) \in R\), either \( r \in \text{sub}(l) \) or \( r \in T(\Sigma, \emptyset) \) is a term in normal form. Permitting terms in \( T(\Sigma, \emptyset) \) on the right-hand side follows [8].

**Example 2.** The rewriting system \( R_{DY} \) over \( \Sigma_{DY} \), formalizing the usual capabilities of the Dolev-Yao intruder, is given by

\[
R_{DY} = \left\{ \pi_1((x, y)) \to x, \pi_2((x, y)) \to y, \{x\}_y^s \to x, \{x\}_{yab} \to x ^{s_{-1}} \right\}.
\]

The rewriting system \( R_p \) represents a Dolev-Yao attacker in the presence of symmetric encryption satisfying the prefix property and is given by \( R_p = R_{DY} \cup \{ \pi_1((x,y))_y^s \to \{x\}_y^s \} \). \( R_{DY} \) and \( R_p \) are convergent rewriting systems. \( R_{DY} \) is also subterm convergent; note, however, that \( R_p \) is not. For readability, we write \( \approx_{DY} \) and \( \approx'_{DY} \) instead of \( \approx_{R_{DY}} \) and \( \approx_{R_p} \), respectively.

Our definitions of deduction and static equivalence differ slightly from those introduced in [4] and used, e.g., in \([2,3,19]\). However, they are equivalent to the original ones (in particular, our definition of deduction is analogous to the characterization provided by Proposition 1 of \([2]\)) and fit our methods better.

**Definition 1.** Given a frame \( \phi \), a term \( t \in T(\Sigma, \text{Name}) \), and an equational theory \( \approx \), we say that \( t \) is deducible from \( \phi \) under \( \approx \), and write \( \phi \vdash \approx t \), if there is \( t' \in \sigma[\phi] \) such that \( t' \approx t \).

The equational theories \( \approx \) that we use are those generated by rewriting systems \( R \); thus, we write \( \phi \vdash_R t \) instead of \( \phi \vdash \approx_R t \).

**Definition 2.** Given two frames \( \phi = \nu \sigma \) and \( \phi' = \nu' \sigma' \) and an equational theory \( \approx \), we say that \( \phi \) and \( \phi' \) are statically equivalent under \( \approx \), and write \( \phi \approx^s \phi' \), if \( T_\phi = T_{\phi'} \) (i.e., \( \nu = \nu' \) and \( \text{dom}(\sigma) = \text{dom}(\sigma') \)) and, for all \( t, t' \in T_{\phi}, t \sigma \approx t' \sigma \) if and only if \( t \sigma' \approx t' \sigma' \).

The corresponding decision problems are defined as expected.

**Definition 3 (Deduction Problem).** Given a frame \( \phi \), an equational theory \( \approx \), and a term \( t \), does \( \phi \vdash \approx t \) hold?

**Definition 4 (Static Equivalence Problem).** Given frames \( \phi \) and \( \phi' \) and an equational theory \( \approx \), does \( \phi \approx^s \phi' \) hold?

Static equivalence is well-suited for modeling off-line guessing attacks \([1,7,19]\). Suppose that a nonce \( g \) has low entropy, which can happen, for example, if it is a human-picked password. Then, an attacker may choose a relatively small set of bitstrings with a high probability of containing the bitstring represented by \( g \). The attacker then uses each of these bitstrings as his guess for the password. The attack is successful if he can verify which of these guesses is correct. The following definition, in the spirit of \([7,19]\), captures this intuition.
Definition 5. Let $\approx$ be an equational theory, $\phi = v \tilde{n} \cdot \sigma$ be a frame, and $g \in \text{Name}$ be a name. Fix some fresh name $w \in \text{Name} \setminus (\text{sub(ran} (\sigma)) \cup \{g\})$. Let $\tilde{n}' = \tilde{n} \cup \{w\}$, and define $\phi_w = v(\tilde{n}') \cdot (\sigma \cup \{x_{n+1} \mapsto g\})$, $\phi_0 = v(\tilde{n}') \cdot (\sigma \cup \{x_{n+1} \mapsto w\})$.

We say that $\phi$ allows an off-line guessing attack of $g$ under $\approx$ if $\phi_g \neq^* \phi_w$.

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

Intuitively, the attacker’s guess can be seen as a message in the network. The attacker does not know beforehand if his guess is correct, but he can check it if he can distinguish a frame in which $x_{n+1}$ stands for a random name $w$ from a frame in which $x_{n+1}$ stands for the guessed name $g$. Section 4.1 presents an application of static equivalence to the study of off-line guessing attacks.

In order to obtain polynomial complexity bounds for our algorithms, we will work with DAG (directed acyclic graph) representations of terms, as in [2].

Definition 6. Let $t \in T(\Sigma, X)$ be a term. Let $V$ be a set (of vertices), $\lambda: V \rightarrow \Sigma \cup X$ a labeling function, $E \subseteq V \times V \times \mathbb{N}$ a set of edges, and $v \in V$ a vertex.

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

If $t = f(t_1, \ldots, t_n)$, then $(V, \lambda, E, v)$ is a DAG-representation of $t$ if $\lambda(v) = f$, there are edges $(v, v_1, 1), \ldots, (v, v_n, n) \in E$ such that, for each $i \in \{1, \ldots, n\}$, $(V, \lambda, E, v_i)$ is a DAG-representation of $t_i$, and there are no other edges $(v, v', m) \in E$ for any $v' \in V$ and any $m \in \mathbb{N}$.

A tuple $T = (V, \lambda, E)$ is a DAG-forest (i.e., a tree with several roots) if, for all $v \in V$, $(V, \lambda, E, v)$ represents some term $t \in T(\Sigma, X)$. If $T = (V, \lambda, E)$ is a DAG-forest and $v \in V$, we use the following notions: $\text{term}_T(v)$ is the (unique) term represented by $(V, \lambda, E, v)$; $e_T(v)$ is the (only) $v' \in V$ such that $(v, v', i) \in E$; $\text{in}_T(v) = \{w \in V \mid (w, v, i) \in E \text{ for some } i\}$; $\text{out}_T(v) = \{w \in V \mid (v, w, i) \in E \text{ for some } i\}$; $\text{leaves}(T) = \{v \in V \mid \text{out}_T(v) = \emptyset\}$; and $\text{roots}(T) = \{v \in V \mid \text{in}_T(v) = \emptyset\}$. If $T$ has only one root, we may refer to it as a DAG-tree.

Let $T = (V, \lambda, E)$ be a DAG-forest. If $\text{roots}(T) = \{v\}$, we say that $T$ is a DAG-representation of the term $\text{term}_T(v)$. When no confusion can arise, we may abuse notation and use the same symbol for such a DAG-forest and the term represented by it. $T$ is minimal if there are no distinct vertices $v, v' \in V$ such that $\text{term}_T(v) = \text{term}_T(v')$. $T$ is in normal form if, for each vertex $v$ in $T$, $\text{term}_T(v)$ is in normal form. A normal form of $T$ is a DAG-forest $T_{nf}$ such that, for all $v \in \text{roots}(T)$, there is a vertex $v_{nf} \in T_{nf}$ such that $\text{term}_{T}(v) \downarrow = \text{term}_{T_{nf}}(v_{nf})$, and for each $v_{nf} \in \text{roots}(T_{nf})$, there is a $v \in \text{roots}(T)$ such that $\text{term}_T(v) \downarrow = \text{term}_{T_{nf}}(v_{nf})$. The minimal normal form of a DAG-forest is unique up to renaming of vertices.
Example 3. The diagram

\[ \begin{array}{ccc}
\text{pair} & \rightarrow & \text{dec} \\
x & \searrow & 1 \\
\downarrow & & 2 \\
& & y \\
\text{enc} & \rightarrow & \text{sym} \\
\downarrow & & 2 \\
& & x \\
& & h \\
\end{array} \]

depicts a DAG-forest containing DAG-representations of the terms \((x, y)\) and \(\{(h(y))^n\}_x\). Its minimal normal form is shown below.

\[ \begin{array}{ccc}
\text{pair} & \rightarrow & \text{dec} \\
x & \searrow & 1 \\
\downarrow & & 2 \\
& & y \\
\end{array} \]

Our complexity results depend on the rewriting system and are stated in terms of the size of terms and frames. If \(t \in T(\Sigma, \text{Name})\) is a term, then the size \(|t|\) of \(t\) is 1 if \(t \in X\) and \(1 + \sum_{i=1}^n |t_i|\) if \(t = f(t_1, \ldots, t_n)\). If \(\phi = v\bar{n} \sigma\) is a frame, then the size \(|\phi|\) of \(\phi\) is given by \(|\phi| = \sum_{x \in \text{dom}(\sigma)} |x\sigma|\). If \(T = (V, \lambda, E)\) is a DAG-forest, we define \(|T| = |V|\). If \(R\) is a rewriting system, we define \(\text{nvars}(R) = \max_{(l \rightarrow r) \in R} |\text{vars}(l)|\).

3 Decision Procedures for Subterm Convergent Rewriting Systems

Throughout this section we assume fixed a subterm convergent rewriting system \(R\) such that \(\text{nvars}(R) \geq 1\) and a frame \(\phi = v\bar{n} \sigma\) such that \(\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}\) and the terms \(t_i\) are all in normal form. We also fix a set \(\mathcal{T} = \{\tau_1, \ldots, \tau_{\text{nvars}(R)}\}\) of fresh names, i.e., \(\mathcal{T} \cap (\bar{n} \cup \text{sub}(\text{ran}(\sigma))) = \emptyset\).

3.1 Frame Saturation

In this section, we present our frame saturation algorithm. Frame saturation is a common technique for deciding both deduction and static equivalence [2, 8, 16]. Our procedure is less general than those implemented in [8, 16], but it is more efficient for subterm convergent equational theories. The existence of a saturation as defined below is closely related to the property of local stability in [3].

Definition 7. We say that \(t\) is a \(\phi\)-recipe (under \(R\)) for \(t'\) if \(t\) is a \(\phi\)-recipe and \(t\sigma \approx_R t'\). We say that a frame \(\phi_s = v\bar{n} \sigma_s\) is a saturation of \(\phi\) (under \(R\)) if there is a \(\phi\)-recipe for each \(t \in \text{ran}(\sigma_s)\) and, for all \(t \in T_\phi\), \((t\sigma)_s \in \sigma_s[T_\phi]\).
The following simple lemma justifies the usefulness of saturation.

**Lemma 1.** Let \( \phi_s \) be a saturation of \( \phi \) and \( t \in T(\Sigma, \text{Name}) \) be a term. Then, \( \phi \vdash_R t \) if and only if \( t \in \sigma_s[T_{\phi_s}] \).

To begin with, in line 1 of our saturation algorithm (Algorithm 1) we build a forest \( T_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R}) \). \( T_{\phi,R} \) is a minimal DAG-forest containing DAG representations of all terms \( \lambda \sigma_l \), where \( l \) is the left-hand side of some rewrite rule \( (l \rightarrow r) \in R \) and \( \sigma_l; \text{vars}(l) \rightarrow \text{sub}(\text{ran}(\sigma)) \cup \mathcal{T} \) is a substitution. We also use the functions \( \zeta \) and \( \text{rw} \). \( \zeta \) is such that, for each vertex \( v \in V_{\phi,R} \) representing a term \( t \in \text{ran}(\sigma) \cup \mathcal{T} \), \( \zeta(v) \) is a DAG-representation of a \( \phi \)-recipe for \( t \). \( \text{rw} \) is such that, for each vertex \( v \) representing a term \( \lambda \sigma_l \) as described above, \( \text{rw}(v) \) is the (unique) vertex representing \( \sigma_l \). Algorithms for computing \( T_{\phi,R} \), \( \text{rw} \) and \( \zeta \) are given in the Appendix. Lemma 2 summarizes their relevant properties.

**Lemma 2.** The forest \( T_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R}) \) and the functions \( \text{rw} \) and \( \zeta \) are such that:

1. \( T_{\phi,R} \) is minimal, can be obtained in time \( \mathcal{O}(|\phi|^{\text{vars}(R)} \log^2 |\phi|) \), and \( |T_{\phi,R}| \in \mathcal{O}(|\phi|^{\text{vars}(R)}) \);
2. \( \text{rw} \) can be computed in time \( \mathcal{O}(\log |\phi|) \);
3. \( \zeta \) can be computed in time \( \mathcal{O}(\log |\phi|) \); for each \( v \in \text{dom}(\zeta) \), \( |\zeta(v)| \in \mathcal{O}(1) \);
4. for each \( s \in \text{sub}(\text{ran}(\sigma)) \cup \mathcal{T} \), there is an unique \( v \) such that \( \text{term}_{T_{\phi,R}}(v) = s \);
5. for each \( v \in \text{dom}(\text{rw}) \), \( \text{term}_{T_{\phi,R}}(v) \rightarrow_R \text{term}_{T_{\phi,R}}(\text{rw}(v)) \);
6. for each \( t \in \text{ran}(\sigma) \cup \mathcal{T} \), there is \( v \) such that \( \text{term}_{T_{\phi,R}}(v) = t \) and \( v \in \text{dom}(\zeta) \);
7. for each \( v \in \text{dom}(\zeta) \), \( \text{term}_{\zeta(v)}(v) \) is a \( \phi \)-recipe for \( \text{term}_{T_{\phi,R}}(v) \);
8. for each rule \( (l \rightarrow r) \in R \) and each substitution \( \sigma_l; \text{vars}(l) \rightarrow \text{sub}(\text{ran}(\sigma)) \cup \mathcal{T} \), there is an unique \( v \in V_{\phi,R} \cap \text{dom}(\text{rw}) \) such that \( \text{term}_{T_{\phi,R}}(v) = 1 \sigma_l \) and \( \text{term}_{T_{\phi,R}}(\text{rw}(v)) = r \sigma_l \).

Our saturation algorithm traverses the forest \( T_{\phi,R} \) bottom-up. At each vertex \( v \), whenever a recipe for \( \text{term}_{T_{\phi,R}}(v) \) is found, \( v \) is added to the range of \( \zeta \) and \( \zeta(v) \) is a DAG-representation of a \( \phi \)-recipe for that term. A recipe is found if one has recipes \( \zeta(v_1), \ldots, \zeta(v_n) \) for all vertices \( v_i \) which have an incoming edge \( (v, v_i, i) \) from \( v \). If the term represented by \( v \) is an instance of the left-hand side of a rule, then this recipe is also stored under \( \zeta(\text{rw}(v)) \) (note that \( \text{term}_{T_{\phi,R}}(v) \rightarrow_R \text{term}_{T_{\phi,R}}(\text{rw}(v))) \). Thus, throughout the saturation process, the function \( \zeta \) associates each vertex \( v \) in its domain to a DAG-representation of a \( \phi \)-recipe for \( \text{term}_{T_{\phi,R}}(v) \). Whenever we add a vertex \( v \) to the domain of \( \zeta \), we add all vertices \( v' \) with an outgoing edge \( (v', v, i) \) to \( v \) to the list of vertices to visit in the next execution of the visiting loop. At the end of the process, a term \( t \in \text{sub}(\text{ran}(\sigma)) \) is deducible from \( \phi \) if and only if the (unique) vertex representing that term is in the domain of \( \zeta \).

At the same time, the algorithm stores the functions \( \sigma_s \) and \( \zeta_s \). \( \sigma_s \) is such that \( \phi_s = \psi \sigma_s \) is a saturation of \( \phi \), and \( \zeta_s \) is such that \( \text{dom}(\zeta_s) = \text{dom}(\sigma_s) \) and, for each \( x \in \text{dom}(\sigma_s) \), \( \zeta_s(x) \) is a DAG-representation of a \( \phi \)-recipe for \( x \sigma_s \).
Furtheing our abuse of notation we use the symbol $\zeta$ as the substitution that assigns, to each $x \in \text{dom}(\zeta_s)$, the term represented by (the DAG-forest) $\zeta_s(x)$. In this case we use postfix notation, i.e., we write $x_{\zeta_s}$.

We state the algorithm’s properties and complexity in Lemma 3.

**Algorithm 1 (Saturating a Frame)**

**Input:** a frame $\phi = v_{\tilde{n}},\sigma$, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$

**Output:** a saturation $\phi_s = v_{\tilde{n}}.\sigma_s$ of $\phi$ and a function $\zeta_s$

1: compute $T_{\phi,R} = (V_{\phi,R},\lambda_{\phi,R},E_{\phi,R}),rw,\zeta$

2: $\zeta_s \leftarrow \{x \mapsto (\{v_x\},\{v_x \mapsto x\},\emptyset) \mid x \in \text{dom}(\sigma)\}$, where the $v_x$ are such that $\text{term}_{\tau_{\phi,R}}(v_x) = x\sigma$

3: $\sigma_s \leftarrow \sigma$

4: visitnext $\leftarrow \text{leaves}(T_{\phi,R}) \cup (\bigcup_{v \in \text{dom}(\zeta)}\text{in}_{\tau_{\phi,R}}(v))$, visitnext $\leftarrow \emptyset$

5: while visitnow $\neq \emptyset$

6: for all $v \in \text{visitnow}$

7: if $\lambda(v) \in X \setminus \tilde{n}$ and $v \notin \text{dom}(\zeta)$ then

8: $\zeta \leftarrow \zeta \cup \{v \mapsto (v,\{v \mapsto \lambda(v)\},\emptyset)\}$

9: visitnext $\leftarrow \text{visitnext} \cup \text{in}_{\tau_{\phi,R}}(v)$

10: if out$_{\tau_{\phi,R}}(v) \subseteq \text{dom}(\zeta)$ and $v \notin \text{dom}(\zeta)$ then

11: let $(V_i,\lambda,\xi_i) = (\zeta(e_i,\tau_{\phi,R}(v)))$ for $i \in \{1,\ldots,\text{ar}(\lambda_{\phi,R}(v))\}$

12: $\zeta_i \leftarrow \zeta \cup \{v \mapsto (v \cup \bigcup_{i=1}^{\text{ar}(\lambda_{\phi,R}(v))}V_i, \lambda_{\phi,R}(v),\xi_i) \cup \bigcup_{i=1}^{\text{ar}(\lambda_{\phi,R}(v))}(\lambda_{\phi,R}(v),\xi_i)\}$

13: if $v \in \text{dom}(\lambda_{\phi,s})$ and $\lambda_{\phi,s}(v) \notin \text{dom}(\zeta)$ then

14: $\zeta \leftarrow \zeta \cup \{\lambda(v) \mapsto \lambda(v)\}$

15: if $\text{term}_{\tau_{\phi,R}}(\lambda_{\phi,s}(v)) \in \text{sub}(\text{run}(\sigma))$

16: then choose $x \in \text{Var} \setminus \text{dom}(\sigma)$

17: $\sigma_s \leftarrow \sigma_s \cup \{x \mapsto \text{term}_{\tau_{\phi,R}}(\lambda_{\phi,s}(v))\}$

18: $\zeta_s \leftarrow \zeta_s \cup \{x \mapsto \zeta(\lambda_{\phi,s}(v))\}$

19: visitnext $\leftarrow \text{visitnext} \cup \text{in}_{\tau_{\phi,R}}(\lambda_{\phi,s}(v))$

20: else visitnext $\leftarrow \text{visitnext} \cup \text{in}_{\tau_{\phi,R}}(v)$

21: return $\zeta_s, \phi_s = v_{\tilde{n}}.\sigma_s$

**Lemma 3.** Algorithm 1 terminates in time $O(|\phi|^{\text{vars}(R)} \log^2 |\phi|)$.

$\phi_s$ is a saturation of $\phi$ (under $R$), $\text{dom}(\zeta_s) = \text{dom}(\sigma_s)$, and, for each $x \in \text{dom}(\sigma_s)$, $\zeta_s(x) \in T_\phi$ and $\zeta_s(x)$ is a DAG-representation of a $\phi$-recipe for $x\sigma_s$ with size $|\zeta_s(x)| \in O(|\phi|)$.

For each $v \in \text{dom}(\zeta)$, there is a $\phi_s$-recipe $t$ for $\text{term}_{\tau_{\phi,R}}(v)$ such that $\zeta(v) = t\zeta_s$ is a $\phi$-recipe for $\text{term}_{\tau_{\phi,R}}(v)$. If $\text{term}_{\tau_{\phi,R}}(v) \in \sigma_s[T_{\phi_s}]$, then $v \in \text{dom}(\zeta)$.

### 3.2 Decision Procedure for Deduction

In light of Lemma 1, to solve the deduction problem under $R$ for a frame $\phi$ and a term $t$ it suffices to compute $t_{\phi,R}$ and the saturated frame $\phi_s = v_{\tilde{n}}.\sigma_s$ (using Algorithm 1) and then decide whether $t_{\phi,R} \in \sigma_s[T_{\phi_s}]$. In the Appendix we see...
that the complexities of these three computations are, respectively, $O(|t| \log^2 |t|)$, $O(|\nu|^{\nuars(R)} \log^2 |\nu|)$ and $O(|t||\phi|^2)$. Theorem 1 sums up these observations.

**Theorem 1.** Given a frame $\nu$ and a term $t$, the complexity of deciding whether $\phi \vdash_R t$ is at most $O(|t| \log^2 |t| + |\nu||\phi|^2 + |\nu|^{\nuars(R)} \log^2 |\phi|)$.

### 3.3 Decision Procedure for Static Equivalence

Throughout this section, $T_{\phi,R}$ is as described in the previous sections, $\phi_s$ and $\zeta_s$ are as output by Algorithm 1, and $\phi = \nu \phi' \sigma$ is a frame such that $\text{dom}(\sigma') = \text{dom}(\sigma) = \{x_1, \ldots, x_n\}$. We assume that $\text{dom}(\sigma_s) = \{x_1, \ldots, x_m\}$, and that $\sigma_s$ is an extension of $\sigma$. $T$ will be used as in the previous section.

Algorithm 2 summarizes our procedure for deciding static equivalence. Note that some of the operations performed by this algorithm must in fact use the DAG-representation of terms to ensure polynomial-time complexity. For simplicity, we defer the exposition of such details to the Appendix.

**Algorithm 2 (Decision Procedure for $\approx_R^*$)**

**Input:** two frames $\phi = \nu \phi' \sigma$ and $\phi' = \nu \phi' \sigma'$ such that $\text{dom}(\sigma) = \text{dom}(\sigma') = \{x_1, \ldots, x_n\}$

**Output:** true if $\phi \approx_R^* \phi'$, false otherwise

1: compute $T_{\phi,R}, \zeta, \text{rw}$, $\zeta_s$ and $\phi_s$
2: if $\phi = \nu \phi' \sigma$ then
3: choose a bijection $i: \{1, \ldots, m_s\} \to \{1, \ldots, m_s\}$ such that $1 \leq j < k \leq m_s \Rightarrow |x_{i(j)}\sigma_s| \leq |x_{i(k)}\sigma_s|$
4: for each $k \in \{1, \ldots, m_s\}$, let $\phi_{s,k} = \nu \zeta_s \{x_{i(1)} \mapsto x_{i(1)}\sigma_s, \ldots, x_{i(k)} \mapsto x_{i(k)}\sigma_s\}$
5: for all $k \in \{1, \ldots, m_s\}$
6: if $x_{i(k)}\sigma_s \in \sigma_s[T_{\phi_s,k-1}]$ then
7: choose $t \in T_{\phi_s,k-1}$ such that $x_{i(k)}\sigma_s = t\sigma_s$
8: if $x_{i(k)}\zeta_s\sigma' \not\approx_R T_{\phi_s,k-1} \zeta_s \sigma'$ then return false
9: for all $v \in \text{dom}(\text{rw})$
10: if $|v| < |\nu|^{\nuars(R)} |\phi'|^2$ then return false
11: return true
12: Repeat lines 1–11, exchanging $\phi$ and $\phi'$

The first loop (lines 5–8) tests whether syntactical equality between terms yielded by two distinct $\phi$-recipes under $\phi$ implies that these two recipes yield equationally equal terms under $\phi'$. This test can be done in time $O((|\phi| + |\phi'|)^3)$. The condition in lines 9–11 tests whether there is a pair of recipes representing instances of the left and right-hand sides of a rule under $\phi$ but that do not represent equal terms (modulo $R$) under $\phi'$. This test takes time $O((|\phi| + |\phi'|)^{\nuars(R)+1} \log^2(|\phi| + |\phi'|))$.

If either of the two loops outputs false the two frames are not statically equivalent. Otherwise, we conclude that all equalities (between recipes, modulo
Algorithm 2 decides whether $\phi \equiv_R \phi'$ in time

$$O(((|\phi| + |\phi'|)^3 + (|\phi| + |\phi'|)^{nvars(R)} + 1 \log^2(|\phi| + |\phi'|)).$$

4 The Prefix Theory

We now investigate how our methods can be extended to deal with theories resulting from other convergent rewriting systems. In particular, we examine the system $R_p$ introduced in Example 2 for representing symmetric encryption with the prefix property. Encryption modes designed to encrypt large messages using block ciphers often have the prefix property, namely ECB and CBC. Although the decidability of this theory has been studied [14], we are not aware of polynomial-time results for static equivalence.

As before, we assume here that $\phi = v \pi_\sigma$ is a frame, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, and fix a set $T = \{\tau_1, \tau_2, \tau_3\}$ of fresh names (note that $nvars(R_p) = 3$).

We begin by defining $p$-subterms, which extend the usual notion of subterm.

**Definition 8.** Let $t \in T(\Sigma, \text{Name})$. We define the set of $p$-subterms of $t$ inductively as follows:

- if $t \in \text{Name}$, then $sub_p(t) = \{t\}$;
- if $t = ((t_1, t_2))^{(s)}_{t_3}$ for some $t_1, t_2, t_3 \in T(\Sigma, \text{Name})$, then $sub_p(t) = \{t, t_3\} \cup sub_p((t_1, t_2));$
- if $t = f(t_1, \ldots, t_n)$ for some $f$ and some $t_1, \ldots, t_n$ and $t \neq ((t'_1, t'_2))^{(s)}_{t_3}$ for all $t'_1, t'_2, t'_3 \in T(\Sigma, \text{Name})$, then $sub_p(t) = \{t\} \cup \bigcup_{i=1}^n sub_p(t_i)$.

We extend this definition to sets of terms as usual.

Our algorithms for the prefix theory use a forest $T_{\phi, R}$ analogous to the tree $T_{\phi, R}$ used for subterm convergent theories. The construction process is also similar. We first compute the substitution $\sigma_{df} = \{x_1 \mapsto t_1 \downarrow, \ldots, x_n \mapsto t_n \downarrow\}$. Then, we build a minimal DAG-forest $T_{\phi, R}$ containing DAG-representations of all terms $lt_i$, where $l$ is the left-hand side of a rewrite rule $(l \rightarrow r) \in R$ and $\sigma; \text{vars}(l) \rightarrow sub_p(\text{ran}(\sigma)) \cup Y$. The functions $rw_\phi$ and $\zeta_\phi$ are defined analogously to $rw$ and $\zeta$ in the tree $T_{\phi, R}$. $T_{\phi, R}$ satisfies properties analogous to those given in Lemma 2, after replacing some critical instances of sub by sub$_p$. A summary of these properties and details of the construction can be found in the Appendix.

Let Algorithm 1$_p$ be Algorithm 1 after replacing $R$ by $R_p$, $T_{\phi, R}$ by $T_{\phi, p}$, and sub by sub$_p$ in line 15. Algorithm 1$_p$ computes a saturated frame under $R_p$. Our algorithms for deduction and static equivalence also work for the prefix theory. We summarize these results in the following theorems. Note, in our complexity analysis, that $nvars(R_p) = 3$ — that is, the complexity of our algorithms is not affected by the fact that $R_p$ is not subterm convergent.
Lemma 4. Algorithm 1 terminates in time $O(|\phi|^3 \log^2 |\phi|)$.

φₙ is a saturation of φ (under $R_\phi$), $\text{dom}(\zeta_\phi) = \text{dom}(\sigma_\phi)$, and, if $x \in \text{dom}(\zeta_\phi)$, then $x\zeta_\phi$ is a DAG-representation of a φ-recipe for $x\sigma_\phi$ with size $|x\zeta_\phi| \in O(|\phi|)$.

For each $v \in \text{dom}(\zeta)$, there is a $\phi_\zeta$-recipe $t$ for $\text{term}_{\tau_\phi}(v)$ such that $\zeta(v) = t\zeta_\phi$ is a φ-recipe for $\text{term}_{\tau_\phi}(v)$. If $\text{term}_{\tau_\phi}(v) \in \sigma_\phi[T_{\phi_\phi}]$, then $v \in \text{dom}(\zeta)$.

Theorem 3. Given a frame φ and a term $t$, the complexity of deciding whether $\phi \vdash_{R_\phi} t$ is at most $O((|t| \log^2 |t| + |t| |\phi|^2 + |\phi|^4 \log^2 |\phi|))$.

Theorem 4. Algorithm 2 decides whether $\phi \approx^*_{R_\phi} \phi'$ in time

$O((|\phi| + |\phi'|)^4 \log^2(|\phi| + |\phi'|))$.

4.1 Off-line Guessing Attacks on a Version of Kerberos

We now present multiple off-line guessing attacks on a version of Kerberos. Most of our attacks rely on the prefix property. Kerberos is known to be insecure against off-line guessing attacks — e.g., [23] describes an attack relying only on standard properties of symmetric encryption that is also captured by our model. However, our formal analysis of its security when implemented with a symmetric encryption scheme satisfying the prefix property is novel.

Kerberos Variant. The version of Kerberos we consider is adapted from [12]. We present a short description of the protocol, in standard notation.

1. $A \rightarrow \text{KAS}: \langle A, \text{KAS}, T_1 \rangle$
2. $\text{KAS} \rightarrow A: \langle \{\{\langle A, \text{TGS}, K_{A,TGS}, T_2 \rangle \}_s\}^*_{K_{A,TGS}, \text{TGS}, T_2}, \{K_{A,TGS}, \text{TGS}, T_2\} \}^*_{K_A, \text{KAS}}$
3. $A \rightarrow \text{TGS}: \langle \{\{\langle A, \text{TGS}, K_{A,TGS}, T_2 \rangle \}_s\}^*_{K_{A,TGS}, \text{TGS}, T_2}, \{A, T_3\} \}^*_{K_A, \text{TGS}}$
4. $\text{TGS} \rightarrow A: \langle \{K_{A,B}, B, T_4, \{\{A, B, K_{A,B}, T_3\} \}_s\}^*_{K_{A,B}, \text{TGS}}, \{\{A, T_5\} \}^*_{K_A, \text{TGS}}$
5. $A \rightarrow B: \langle \{A, B, K_{A,B}, T_4\} \}^*_{B, \text{TGS}}, \langle\{\{A, T_5\} \}^*_{K_A, \text{TGS}}$
6. $B \rightarrow A: \langle T_5 \}^*_{K_{A,B}}$

$A$ is a name (e.g., of a client of a single-sign-on service). $\text{KAS}$ is the Kerberos authentication server. $\text{TGS}$ is the ticket-granting server. $B$ is some service provider. $K_{A,\text{KAS}}$ (respectively $K_{A,TGS}$, $K_{B,TGS}$) is a long-term key shared between $A$ and $\text{KAS}$ (respectively between $\text{KAS}$ and $\text{TGS}$ and between $B$ and $\text{TGS}$). $K_{A,TGS}$ (respectively $K_{A,B}$) is a short-term key shared between $A$ and $\text{TGS}$ (respectively between $A$ and $B$), freshly generated for each session by $\text{KAS}$ (respectively by $\text{TGS}$). Finally, $T_1, \ldots, T_5$ are timestamps. Note that for our purposes, it suffices to view them as freshly generated nonces, even if this representation is imprecise.

The only difference between the protocol presented here and the model of Kerberos (version IV) presented in [12] is that, in the term sent in step 2, the encryption $\{\{A, TGS, K_{A,TGS}, T_2\} \}^*_{K_{A,TGS}}$ is the first (instead of the last) term in the encrypted tuple. Although this may appear to be a small and harmless change, it gives rise to guessing attacks if a symmetric encryption scheme with
We consider an attacker who eavesdrops on the double encryption used in version IV of the protocol. Version V eliminates the double encryption and thereby prevents this attack.

The signature $\Sigma^{KEK}$ we use is obtained by simply adding the set $\Sigma_0 = \{A, B, KAS, TGS\}$ of agent names to the signature $\Sigma^{DY}$. We assume that $\{T_i \mid i \in \mathbb{N}\} \subseteq \text{Name}$. We also represent the long-term keys $K_{A,KAS}, K_{KAS,TGS} \in \text{Name}$ and the short-term keys $K_{A,TGS}, K'_{A,TGS} \in \text{Name}$ (corresponding to the keys generated by KAS for use between A and TGS in two distinct sessions) as names.

The Attacker’s Knowledge We consider an attacker who eavesdrops on two different protocol sessions, both executed between an agent A and the server KAS. For simplicity, we assume that the attacker stores only the second message of each session. This is enough to present our off-line guessing attacks. We represent the attacker’s knowledge by the frame $\phi = v_n, \sigma$, where $\tilde{n} = \left\{K_{A,TGS}, K'_{A,TGS}, K_{A,KAS}, K_{KAS,TGS}, T_1, T_2\right\}$ and

$$\sigma = \{x_0 \mapsto \left\{\left\langle\{A, TGS, K_{A,TGS}, T_1\}\right\rangle^{s}_{K_{A,TGS},TGS}, K_{A,TGS}, TGS, T_1\right\}^{s}_{K_{A,KAS}}, \%
$$

$$x_1 \mapsto \left\{\left\langle\{A, TGS, K'_{A,TGS}, T_2\}\right\rangle^{s}_{K'_{A,TGS},TGS}, K'_{A,TGS}, TGS, T_2\right\}^{s}_{K_{A,KAS}}\}\}.$$

We are interested in determining whether this frame allows an off-line guessing attack of $K_{A,KAS}$. This is a (potentially) weak key, since it is often chosen by human users or derived from such a key. We take $g = K_{A,KAS}$ and $w = a_0$ and, throughout our example, we work with the frames $\phi_g = v\tilde{n}', \sigma_g$ and $\phi_w = v\tilde{n}', \sigma_w$, where $\tilde{n}' = \tilde{n} \cup \{a_0\}$, $\sigma_g = \sigma \cup \{x_2 \mapsto K_{A,KAS}\}$, and $\sigma_w = \sigma \cup \{x_2 \mapsto a_0\}$. In the Appendix, we present the results of saturating the frames $\phi_w$ and $\phi_g$ in Tables 2 and 3.

Off-line Guessing Attacks to Kerberos It is clear that $\phi_w \not\approx_{R_{\phi}} \phi_g$. Thus, Kerberos allows an off-line-guessing attack of $K_{A,KAS}$. In fact, an attacker has multiple pairs of recipes $t, t' \in T_{\phi_g}$ that he can use to validate his guess; we present a few illustrative examples in Table 1. Note that, of the four attacks presented, all but Attack 1 rely on the prefix property of the encryption scheme. Only Attack 4 relies on the fact that we use version IV instead of version V and exchange the order of the messages of the original Kerberos protocol.

How feasible are these attacks in practice? First of all, CBC encryption mode uses a random initialization vector. To prevent Attack 4 it is enough that the initialization vectors used in the two messages are not the same (even if they are public). This can be captured in our model by representing symmetric encryption as a function $\text{enc}_{\text{sym}}$ with three arguments (an initialization vector, the message, and the key) and writing the rewriting rule representing the prefix property as

$$\pi_1(\text{enc}_{\text{sym}}(IV, \langle M_1, M_2, \rangle, K)) \rightarrow \text{enc}_{\text{sym}}(IV, M_1, K).$$

Attack 4 does not arise in this model.
Furthermore, consider the recipes $t$ and $t'$ given in Attack 4. We have $t \sigma_g = t' \sigma_g = \{(A, TGS)\}_{\text{KAS}, TGS}$. However, in practice, all that the attacker can do is obtain an encryption of the first block of the plaintext, and attack is only feasible if the first encrypted block is equal in both messages. This depends on the length of the agent name, the length of the encryption block, whether the agent name is padded to fill at least one block and how this padding is performed.

(FLAG: ) Finally, note that Attack 2 (respectively 3) is only feasible if the encryption of the first three (respectively two) elements of the tuple occupies disjoint encryption blocks from the encryption of the last one (respectively two) elements.

The relevance of details such as initialization vectors, block length, and padding techniques in the study of off-line guessing attacks has been previously pointed out [13]. We believe that it is an important challenge for symbolic methods to provide methods to reason about these kinds of possible weaknesses.

Table 1. (Sample) Off-line Guessing Attacks to Kerberos

<table>
<thead>
<tr>
<th>Attack</th>
<th>$t$</th>
<th>$t'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\pi_2(\pi_1(x_0)^{s_2^{-1}})$</td>
<td>TGS</td>
</tr>
<tr>
<td>2</td>
<td>${(\pi_1(x_0)^{s_2^{-1}}, \pi_2(x_0)^{s_2^{-1}})}_{s_2}$</td>
<td>$x_0$</td>
</tr>
<tr>
<td>3</td>
<td>$\pi_1(\pi_1(\pi_1(x_0)^{s_2^{-1}}, \pi_2(x_0)^{s_2^{-1}}, TGS))$</td>
<td>$\pi_1(\pi_1(x_0)^{s_2^{-1}})$</td>
</tr>
<tr>
<td>4</td>
<td>$\pi_1(\pi_1(\pi_1(\pi_1(x_0)^{s_2^{-1}}))))$</td>
<td>$\pi_1(\pi_1(\pi_1(\pi_1(x_1)^{s_2^{-1}}))))$</td>
</tr>
</tbody>
</table>

5 Related Work and Conclusion

Our algorithms compare favorably to previously existing algorithms. [2] presents the first proof that deduction and static equivalence under subterm convergent equational theories can be decided in polynomial-time. However, efficiency is not their main concern and it is not surprising that our algorithms achieve a better asymptotic complexity: for example, for the theory $\approx_{DY}$, the complexities of our algorithms are $O(|t| \log |\phi| + |\phi|^2 |t| + |\phi|^2 \log^2 |\phi|)$ and $O((|\phi| + |\phi'|)^3 \log^2 (|\phi| + |\phi'|))$ for deduction and static equivalence (respectively), whereas our best estimates for the complexity of the algorithms in [2] are $O(|\phi|^{10} + |\phi'|^2 |t| + |t|^4)$ and $O(|\phi| + |\phi'|^{15})$ for the same problems.

The complexity of the YAPA tool [8] is not polynomial, as it uses a straightforward representation of terms, as opposed to DAGs. Furthermore, our saturation technique is also more efficient: in YAPA, for each $(n, p, q)$-decomposition of the left-hand side of a rewrite rule and each assignment of the first $n + p$ parameters to recipes in the frame, it may be necessary to compute the normal form of a term $t$. We are not aware of any general algorithms for this task achieving a better complexity than $O(|t|^{14})$ (see discussion below). If we denote by $Y(R)$ the greatest value of $n + p$ for all $(n, p, q)$-decompositions of rewriting rules in $R$, then
we obtain a complexity of $O(|\phi|^\text{Y}(R)+4)$ for YAPA’s saturation procedure. Note that $\text{nvars}(R) \leq \text{Y}(R)$ in general. For the rewriting system $R_{DY}$, for example, we thus obtain a complexity of $O(|\phi|^7)$ for YAPA’s saturation procedure and $O(|\phi|^2 \log^2 |\phi|)$ for our. Note that the exact implementation of DAGs and the algorithms to compute normal forms may affect these estimates. It may also be possible to provide better bounds on the number of recipes for which we need to perform this reduction to a normal form.

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

Finally, we note that all the algorithms discussed require deciding the equality of two terms $t, t'$ under the equational theory. Our algorithms rely on the subterm convergence of the rewriting system to perform this task with complexity $O((|t| + |t'|) \log(|t| + |t'|))$. For more general rewriting systems we are not aware of any algorithm improving the complexity of $O((|t| + |t'|)^3)$ achieved in [2].

As future work we plan to implement our algorithms. We also plan to study how our approach can be adapted to other equational theories. The simplicity of extending our methods to the prefix theory suggests that it may be feasible to generalize our approach for a much larger class of equational theories, possibly improving upon existing complexity results for the two decision problems when such results exist.

References

1. Abadi, M., M. Baudet and B. Warinschi, Guessing Attacks and the Computational Soundness of Static Equivalence, Lecture Notes in Computer Science 3921, 398–412


6 Appendix

For readability and ease of access we separate the Appendix in sections, following the numbering in the body of the paper.

**Section 3.1**

Proof (Lemma 1). Note first that $\phi \vdash_R t$ if and only if $\phi \vdash t\downarrow$.

Now, if $t \in \sigma_s[T_\phi]$, then there exists $t' \in T_\phi$ such that $t'\sigma_s = t$. By a simple inductive argument on $t'$, we conclude that $t'\downarrow = t$. Since $t'\downarrow \in T_\phi$, we conclude that $\phi \vdash t$. Conversely, suppose that $\phi \vdash t$. Then there is a $t' \in \sigma[T_\phi]$ such that $t'\downarrow = t$. By the definition of saturation, $t \in \sigma_s[T_\phi]$.

\[\square\]

The addressing tree. Consider a minimal DAG-forest $T = (V, \lambda, E)$ whose vertices represent terms in $T(\Sigma, X)$. Given $x \in X$, we want to be able to efficiently find (or determine that there is no) vertex $v \in V$ such that $\lambda(v) = x$. Similarly, given $f \in \Sigma_n$ and $v_1, \ldots, v_n \in V$, we want to be able to efficiently find or determine that there is no $v \in V$ such that $\lambda(v) = f$ and $e_i, \tau(v) = v_i$ for each $i \in \{1, \ldots, n\}$.

We assume that there is an invertible function $\text{encoding}(v) : X \cup \{(f, v_1, \ldots, v_n) \mid f \in \Sigma_n, v_1, \ldots, v_n \in V\} \to \{0, 1\}^*$ such that both it and its inverse are computable with linear complexity. Note that we can simply choose, for example, a computer’s internal representation of the vector. We abuse notation by writing, for $v \in V$, $\text{encoding}(v) = \lambda(v)$ if $l(v) \in X$ and $\text{encoding}(v) = (f, v_1, \ldots, v_n)$ if $l(v) = f$ and $e_i, \tau(v) = v_i$ for each $i \in \{1, \ldots, n\}$.

To solve these problems we will use a particular data structure, which we will call addressing tree. It can be described as a DAG-tree (a DAG-forest with a single root) whose vertices represent terms in $T(\{\text{nextbit}\}, V \cup \{\bot\})$, where $\text{ar}(\text{nextbit}) = 2$. Intuitively, if a leaf is labeled with a vertex $v$, then the sequence of edges leading from the root to this leaf represent the bitstring $\text{encoding}(v)$.
(with some convention for translating the edge numbers 1 and 2 to the bits 0 and 1). Note that, since \( T \) is minimal, \( v \) is the only element of \( V \) with this encoding. Otherwise, if a leaf is labeled with \( \bot \), then the bitstring represented by the sequence of edges leading to this leaf is not a prefix of encoding(v) for any \( v \in V \). The construction process is merely adding to this tree a branch representing each vertex in \( V \).

This addressing tree requires an amount of space in \( O(|T| \log^2 |T|) \), since there are \( O(|T|) \) branches, each with \( O(\log |T|) \) nodes, and each containing pointers whose size may be up to \( O(\log |T|) \). Similarly, we conclude that this tree can be obtained from \( T \) in time \( O(|T| \log^2 |T|) \) and that, given this tree, the problem above can be solved in time \( O(\log^2 |T|) \). It is important to note that, if one adds a vertex to \( T \), updating the addressing tree takes only \( O(\log |T|) \) time (assuming that the space allocated for each node is enough to store the names of all vertices in the final tree).

A possible implementation consists of a set of objects, each representing a node in the tree and containing three fields: one for storing a vertex and two for the pointers to the children nodes. For simplicity, we will omit mentions to this addressing tree when describing our algorithms and use it solely for our complexity analysis.

\[\text{Algorithm 3 (Computing a Minimal Normal Form)}\]

\textbf{Input:} a DAG-forest \( T = (V, \lambda, E) \)
\textbf{Output:} a minimal normal form \( T_{\min} = (V_{\min}, \lambda_{\min}, E_{\min}) \) of \( T \) and a function \( \text{min}: \text{roots}(T) \rightarrow V_{\min} \)

1. visitnow \( \leftarrow \text{leaves}(T) \), visitnext \( \leftarrow \emptyset \), min \( \leftarrow \emptyset \)
2. \( T_{\min} = (V_{\min}, \lambda_{\min}, E_{\min}) \leftarrow (\emptyset, \emptyset, \emptyset) \)
3. while visitnow \( \neq \emptyset \)
4. for all \( v \in \text{visitnow} \)
5. if \( \text{out}_T(v) \subseteq \text{dom}(\text{min}) \)
6. then visitnext = visitnext \( \cup \text{in}_T(v) \)
7. if there is \( v_{\min} \in V_{\min} \) such that \( \lambda_{\min}(v_{\min}) = \lambda(v) \)
8. and \( e_{i,T_{\min}}(v_{\min}) = \text{min}(e_{i,T}(v)) \) for all \( i \)
9. then min \( \leftarrow \text{min} \cup \{ \text{term}_T(v) \mapsto \text{term}_{T_{\min}}(v_{\min}) \} \)
10. else
11. if there are \( (l \rightarrow r) \in R \) and \( \sigma_l: \text{vars}(l) \rightarrow T(\Sigma, X) \) s.t.
12. \( \left( \lambda(v) \right)\left\{ \text{term}_{T_{\min}}(\text{min}(e_{1,T_{\min}}(v))), \ldots, \right. \)
13. \( \left. \text{term}_{T_{\min}}(\text{min}(e_{\text{ar}(\lambda(v)),T_{\min}}(v)))) \right\} = l\sigma_l \)
14. then min = min \( \cup \{ v \mapsto v_{\min} \} \), where \( \text{term}_{T_{\min}}(v_r) = r\sigma_l \)
15. else
16. choose \( v_{\min} \notin V_{\min} \)
17. min \( \leftarrow \text{min} \cup \{ v \mapsto v_{\min} \} \)
18. \( V_{\min} \leftarrow V_{\min} \cup \{ v_{\min} \} \)
19. \( \lambda_{\min} \leftarrow l_{\min} \cup \{ v_{\min} \mapsto \lambda(v) \} \)
20. \( E_{\min} \leftarrow E_{\min} \cup \{ (v_{\min}, \text{min}(e_{i,T}(v)), i) \mid 1 \leq i \leq \text{ar}(\lambda(v)) \} \)
Given a DAG-forest $T$, Algorithm 3 computes a minimal normal form $T_{\min}$ = $(V_{\min}, \lambda_{\min}, E_{\min})$ of $T$ in time $O(|T| \log^2 |T|)$, and the output function $\min: \text{roots}(T) \rightarrow V_{\min}$ is such that, for all $v \in \text{roots}(T)$, $(\text{term}_T(v))_{\downarrow} = \text{term}_{T_{\min}}(\min(v))$.

Proof. We prove that, throughout all executions of the loop in lines 3-18, two loop invariants are preserved: (A) the forest $T_{\min}$ is minimal, and (B), for each $v \in \text{dom}(\min)$, term$_{T_{\min}}(\min(v)) = (\text{term}_T(v))_{\downarrow}$. The proof is by induction on the number of executions of the loop. Clearly, the initial definitions of $\min$ and $T_{\min}$ on lines 1 and 2 satisfy these properties.

We prove that these properties are preserved by each execution of the inner loop (lines 4-16), that is, by each visit to a node in $V$. If out$_T(v) \not\subseteq \text{dom}(\min)$, then neither $\min$ nor $T_{\min}$ are changed. Otherwise, by the induction hypothesis, we have term$_{T_{\min}}(\min(e_{i,v}(v))) = (\text{term}_T(e_{i,v}(v)))_{\downarrow}$ for each $i$.

If the condition on line 7 holds, then no other vertices are added to $T_{\min}$, so that (A) remains valid. We also have term$_{T_{\min}}(v_{\min}) = (\text{term}_T(v))_{\downarrow}$ (by the induction hypothesis), and thus (B) is also preserved after line 8.

Suppose then the condition on line 7 does not hold. There are two cases: either the condition on line 10 holds (case (1)) or not (case (2)). In case (1), $r\sigma_l$ is a proper subterm of $l\sigma_l$ (otherwise the rewriting system would not be terminating); thus, $r\sigma_l$ is a subterm of term$_{T_{\min}}(\min(e_{i,v}(v)))$ for some $i$. Therefore, there is a vertex $v_r \in V_{\min}$ such that term$_{T_{\min}}(v_r) = r\sigma_l$. We have:

$$\text{term}_T(v) = (\lambda(v))(\text{term}_T(e_{1,v}(v))), \ldots, \text{term}_T(e_{ar(\lambda(v)),T(v)}))$$

$$\approx_R (\lambda(v))((\text{term}_{T_{\min}}(\min(e_{1,v}(v))), \ldots, \text{term}_{T_{\min}}(\min(e_{ar(\lambda(v)),T(v)}))))$$

$$\rightarrow_R r\sigma_l = \text{term}_{T_{\min}}(v_r).$$

By the induction hypothesis, term$_{T_{\min}}(v_r)$ is in normal form. Thus, condition (B) is preserved. Condition (A) is preserved since no vertex is added to $T_{\min}$.

Now, consider the case when the condition on line 10 is false. In this case, we add a vertex $v_{\min}$ to the forest $T_{\min}$ and set $\min(v) = v_{\min}$. After updating $T_{\min}$ in lines 12–16, we have:

$$\text{term}_{T_{\min}}(v_{\min}) = (\lambda(v))(\text{term}_{T_{\min}}(\min(e_{1,v}(v))), \ldots, \text{term}_{T_{\min}}(\min(e_{ar(\lambda(v)),T_{\min} (v)})))$$

$$\approx_R (\lambda(v))((\text{term}_T(e_{1,v}(v))), \ldots, \text{term}_T(e_{ar(\lambda(v)),T(v)}))$$

$$= \text{term}_T(v),$$

where the second equality uses the induction hypothesis. Since the condition on line 10 is false, we conclude that this term is in normal form (note that all its proper subterms are in normal form by the induction hypothesis). Thus, (B) is preserved.
Since the condition on line 7 is also false, it is clear that there is no other vertex \( v'_{\text{min}} \in V'_{\text{min}} \) such that

\[
\text{term}_{\text{min}}(v'_{\text{min}}) = (\lambda(v))(\text{term}_{\text{min}}(\min(e_1, \tau(v))), \ldots, \\
\text{term}_{\text{min}}(\min(e_{\text{ar} \lambda(v)}, \tau(v))))
\]

\[
= \text{term}_{\text{min}}(v_{\text{min}}).
\]

We conclude that (A) is also preserved.

It is easy to see that, in the end of the loop in lines 3–18, \( V \subseteq \text{dom}(\text{min}) \).

At this point of the execution of the algorithm, \( T_{\text{min}} \) is a minimal DAG-forest and \( \text{min}: \text{V} \to V_{\text{min}} \) such that, for each \( v \in V \), \( \text{term}_{T_{\text{min}}}(\text{min}(v)) \equiv (\text{term}_{\tau}(v)) \downarrow \).

After lines 19–20, \( T_{\text{min}} \) is still minimal, and \( \text{term}_{T_{\text{min}}}(v) \) is in normal form for all vertices \( v \in V_{\text{min}} \). It is clear that \( \text{roots}(T_{\text{min}}) \subseteq \text{min}[\text{roots}(T)] \); thus, for all \( v_{\text{min}} \in \text{roots}(T_{\text{min}}) \), there is \( v \in T \) such that \( \text{min}(v) = v_{\text{min}} \), and we have \( \text{term}_{T_{\text{min}}}(v_{\text{min}}) = (\text{term}_{\tau}(v)) \downarrow \).

Finally, for all \( v \in \text{roots}(T) \), \( \text{min}(v) \) a vertex in \( T_{\text{min}} \) such that \( \text{term}_{\text{min}}(v_{\text{min}}) = (\text{term}_{\tau}(v)) \downarrow \).

We conclude that \( T_{\text{min}} \) is a minimal normal-form of \( T \), as desired, and we have already seen that \( \text{min} \) satisfies the lemma.

To estimate the complexity of the algorithm, note first that we can keep an addressing tree for \( T_{\text{min}} \) at every point of the execution of the algorithm (note that \( T_{\text{min}} \) is always minimal) by adding a branch to the addressing tree whenever we add a vertex to \( T_{\text{min}} \). As seen above, this operation has complexity \( O(|T|) \), as is executed at most once per visit to a vertex in \( T \) (i.e., each execution of the loop in lines 4–16).

Consider then a visit to a vertex \( v \). By using our addressing tree, testing the condition on line 7 can be done in time \( O(|T|) \). Testing the condition on line 10 can also be done in time \( O(|T|) \). This is because there is a constant number of rules, each with constant size. Thus, checking whether the term represented by \( v \) matches an instance of a rule involves only checking the labels of a constant number of vertices, and possibly comparing two vertices for equality if the same variable occurs twice in \( l \) (note that, since \( T_{\text{min}} \) is minimal, two vertices \( v \) and \( v' \) represent the same term if and only if \( v = v' \)). These checks take time \( O(|T|) \).

It is clear that any of the operations that may need to be performed (depending on whether these conditions hold) can also be done in logarithmic time. Thus, we obtain a complexity of \( O(|T| \log^2 |T|) \) for each visit to a vertex.

To count the number of visits to vertices, observe that each leaf is visited once and each other vertex \( v \) is visited at most \( |\text{out}_{T}(v)| \) times. Thus, the total number of visits is at most \( |\text{leaves}(T)| + \sum_{v \in V} |\text{out}_{T}(v)| \leq |\text{leaves}(T)| + |E| \in O(|T|) \).

We obtain a total complexity of \( O(|T| \log^2 |T|) \). \( \Box \)

In Algorithm 4 we use the union of disjoint DAG-forests as defined below.

**Definition 9.** Let \( T_1 = (V_1, \lambda_1, E_1) \) and \( T_2 = (V_2, \lambda_2, E_2) \) be DAG-forests such that the \( V_1 \cap V_2 = \emptyset \). We define the union \( T_1 \cup T_2 \) of \( T_1 \) and \( T_2 \) as \( T_1 \cup T_2 = (V_1 \cup V_2, \lambda_1 \cup \lambda_2, E_1 \cup E_2) \). We extend this definition as expected to an arbitrary number of DAG-forests \( T_1, \ldots, T_n \).
Algorithm 4 (Computing $T_{\phi,R}$)

Input: a frame $\phi = v_1, \sigma$, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$

Output: a DAG-forest $T_{\phi,R}$ and the functions $\zeta, \tau$

1: let $T_1, \ldots, T_{n+\text{nvars}(R)}$ be DAG-representations of $t_1, \ldots, t_n, \tau_1, \ldots, \tau_{\text{nvars}(R)}$ with roots $v_1, \ldots, v_{n+\text{nvars}(R)}$
   and disjoint sets of vertices
2: let $T_{\phi} = (V_{\phi}, \lambda_{\phi}, E_{\phi})$ and min be the output of Algorithm 3 on input
   $\bigcup_{i=1}^{n+\text{nvars}(R)} T_i$
3: $T_{\phi,R} = (V_{\phi,R}, \lambda_{\phi,R}, E_{\phi,R}) \leftarrow T_{\phi}$
4:
5: $\zeta \leftarrow \min(v_{n+i}) \mapsto (\{v_{n+i}\}, \{v_{n+i} \mapsto \tau_i\}, \emptyset) \mid i \in \{1, \ldots, \text{nvars}(R)\}$
6: for $i = 1$ to $n$
7: if min($v_i$) $\notin \text{dom}(\zeta)$
8: then $\zeta \leftarrow (\zeta \cup \{v \mapsto (\{v_i\}, \{v_i \mapsto x_i\}, \emptyset)\})$
9: $\text{rw} = \emptyset$
10: $\text{rw} \leftarrow \text{rw} \cup \{v_i \mapsto v_r\}$, where $\text{term}_{\tau_{s,R}}(v_i) = \lambda_{\phi}$ and $\text{term}_{\tau_{s,R}}(v_r) = \text{rw}(v)$

Proof (Lemma 2). $T_{\phi,R}$ contains one vertex for each $s \in \text{sub}(\text{ran}(\sigma)) \cup \{\tau\}$ and at most a constant number of vertices for each $(l \mapsto r) \in R$ and each substitution $\sigma_l: \text{vars}(l) \mapsto \text{sub}(\text{ran}(\sigma)) \cup \{\tau\}$. There are $O(|\phi|^{|\text{nvars}(R)|})$ such substitutions, and we conclude that $|T_{\phi,R}| \in O(|\phi|^{|\text{nvars}(R)|})$. The fact that $T_{\phi,R}$ is minimal is clear because $T_{\phi}$ is minimal and line 12 ensures that $T_{\phi,R}$ is kept minimal whenever we add new vertices.

To obtain our complexity estimate, we first note that each execution of the loop in lines 12–13 takes only time $O((\log^2 |\phi|))$. This is because there is at most a constant number $|l|$ of vertices that may need to be added to the forest. For each of these vertices we need to check whether $T_{\phi,R}$ already contains a vertex representing the same term represented by that term, and we have already seen that our direct addressing tree allows us to do this in time $O((\log^2 |T_{\phi,R}|))$. Since $|T_{\phi,R}| \in O(|\phi|^{|\text{nvars}(R)|})$, this is the same as $O((\log^2 |\phi|))$. Updating the addressing tree can be done with the same complexity. Now, computing the initial forest $T_{\phi}$ can be done in time $O(|T| \log^2 |T|)$ (by Lemma 5), and the loop in lines 12–13 is executed $O(|\phi|^{|\text{nvars}(R)|})$ times; thus, we obtain a total complexity of $O(|\phi|^{|\text{nvars}(R)|} \log^2 |\phi|)$.

To see (2), observe that we can store $\text{rw}(v)$ under some address that can be computed from $v$.

Property (4) is a consequence of (8) and the facts that $T_{\phi,R}$ is minimal and $R$ is not empty.

---

3 One can, for example, implement a vertex $v$ as an object which stores, among other things, whether $v \in \text{dom}(\text{rw})$ and, in this case, $\text{rw}(v)$.
Property (5) is ensured by the instruction on line 13, and properties (6) and (7) are ensured by the initialization of $\zeta$ in lines 5–8.

Property (8) is follows from the loop in lines 11–13.

Lemma 6. Let $t$ be a term and $\phi = v_i.\sigma$ be a frame. Finding (or determining that there is no) $t' \in T_\phi$ such that $t' \sigma = t$ can be done in time $O(|t||\phi|)$.

Proof. Note that $t \in \sigma[T_\phi]$ if and only if either $t \in \text{ran}(\sigma)$, $t \in \text{Name} \setminus \hat{n}$ or $t = f(t_1, \ldots, t_n)$ for some $f \in \Sigma_n$ and some $t_1, \ldots, t_n \in \sigma[T_\phi]$. Thus, at most, we need to check whether $s \in (\text{Name} \setminus \hat{n}) \cup \text{ran}(\sigma)$ for each subterm $s \in \text{sub}(t)$. There are at most $|t|$ such subterms. For each of these subterms, one takes each term $t' \in \text{ran}(\sigma)$ and checks whether $s = t'$. For each $t'$, this check takes at most time $O(|t'|)$. Thus, all checks can be made in $O(\sum_{t \in \text{ran}(\sigma)|t'|}$ - that is, in $O(|\phi|)$. We obtain an overall complexity of $O(|t||\phi|)$.

Lemma 7. Let $R$ be a subterm convergent rewriting system, $(\rightarrow_r) \in R$ be a rule, and $t = f(t_1, \ldots, t_n)$ be a term such that all $t_i$ are in normal form and $t = l_\sigma t_i$ for some $\sigma_i$; $\text{vars}(l) \rightarrow T(\Sigma, \text{Name})$. Then $r \sigma_i$ is in normal form.

Proof. Either $r \sigma_i \in T(\Sigma, \emptyset)$ or $r \sigma_i \in \text{sub}(f(t_1, \ldots, t_n))$. In the first case, $r \sigma_i$ is in normal form by hypothesis. In the second case, $r \sigma_i$ must be a proper subterm of $l_\sigma t_i$ (otherwise the rewriting system would not be convergent). But $l_\sigma t_i = f(t_1, \ldots, t_n)$, and since $t_1, \ldots, t_n$ are in normal form, so are all their subterms. We conclude that all proper subterms of $l_\sigma t_i$ are in normal form and the result follows.

Lemma 8. Let $\phi = v_i.\sigma$ be a frame, $t \in \sigma[T_\phi]$, and $s \in \text{sub}(t)$. Then $s \in \text{sub}(\text{ran}(\sigma))$ or $s \in \sigma[T_\phi]$.

Proof. Let $u \in T_\phi$ be the term such that $u \sigma = t$. The proof is by induction on $u$.

For the base case, either $u \in \text{Name} \setminus \hat{n}$ or $u \in \text{dom}(\sigma)$. If $u \in \text{Name} \setminus \hat{n}$, the result is clear. If $u \in \text{dom}(\sigma)$, then $t \in \text{ran}(s)$, and $s \in \text{sub}(\text{ran}(\sigma))$.

Now let $u = f(u_1, \ldots, u_n)$ for some $u_1, \ldots, u_n \in T_\phi$. Then, we have $t = f(u_1, \ldots, u_n)$. If $s \in \text{sub}(l_t)$, then either $s = t$ or $s \in \text{sub}(u_i)$ for some $i$. If $s = t$, then $s \in \sigma[T_\phi]$. If $s \in \text{sub}(u_i)$ for some $i$, the results holds by the induction hypothesis.

Lemma 9. Let $\phi = v_i.\sigma$ be a frame, $t \in T(\Sigma, \text{Name} \cup \text{Var})$, and $\sigma_1, \sigma_2$ be substitutions such that, for all $x \in \text{vars}(t)$, either $x \sigma_1 = x \sigma_2$ or $x \sigma_1, x \sigma_2 \in \sigma[T_\phi] \setminus \text{sub}(\text{ran}(\sigma))$. Then $\sigma_1 \in \sigma[T_\phi]$ if and only if $l_\sigma \sigma_2 \in \sigma[T_\phi]$.

Proof. The proof is by induction on $t$. The base case is $t \in \text{Var}$. In this case, either $l_\sigma \sigma_1 = l_\sigma \sigma_2$ or both $l_\sigma \sigma_1 \in \sigma[T_\phi]$ and $l_\sigma \sigma_2 \in \sigma[T_\phi]$; in both cases the result holds trivially. If $t \in \text{Name}$, then $l_\sigma \sigma_1 = l_\sigma \sigma_2 = t$, and the result is also trivial.

If $t = f(t_1, \ldots, t_n)$ for some $f \in \Sigma_n$ and some $t_1, \ldots, t_n \in T(\Sigma, \text{Name} \cup \text{Var})$, then $l_\sigma \sigma_1 = f(t_{1,1}, \ldots, t_{1,n})$ and $l_\sigma \sigma_2 = f(t_{2,1}, \ldots, t_{2,n})$.

If $t = f(t_1, \ldots, t_n) \in \sigma[T_\phi]$, then either $t \in \text{ran}(\sigma)$ of $t_i \in \sigma[T_\phi]$ for all $i$, and:
if $t_i \in \sigma[T_\phi]$ for all $i$, then, by the induction hypothesis, $t_i\sigma_2 \in \sigma[T_\phi]$, and $t\sigma_2 \in \sigma[T_\phi]$ as desired;
- if $t\sigma_1 \in \text{ran}(\sigma)$, then for all $x \in \text{vars}(t)$, $x\sigma_1 \in \text{sub}(\text{ran}(\sigma))$; thus, $t\sigma_1 = t\sigma_2$.

Thus, if $t\sigma_1 \in \sigma[T_\phi]$, then $t\sigma_2 \in \sigma[T_\phi]$, and we conclude the demonstration. \qed

Proof (Lemma 3). We prove, for all $t \in T_\phi$, that (A) $(t\sigma)\downarrow \in \sigma_s[T_\phi]$, and (B) if $(t\sigma)\downarrow$ is represented by some vertex $v$ in $T_{\phi,R}$ (that is, if $(t\sigma)\downarrow = \text{term}_{T_{\phi,R}}(v)$ for some $v \in V_{\phi,R}$), then $v \in \text{dom}(\zeta)$ and there is $t' \in T_\phi$ such that $(t'\sigma)\downarrow = (t\sigma)\downarrow$ and $\zeta(v)$ is a DAG-representation of $t'\zeta_s \in T_\phi$.

The proof is by induction on $t$. If $t \in \text{dom}(\sigma)$, then (A) holds due to the initialization of $\zeta_s$ in line 2. (B) holds due to the property (6) in Lemma 2.

If $t \in \text{Name} \setminus \bar{n}$, then (A) trivially holds, and (B) holds because in the first execution of the loop 5–20 the algorithm visits the leaves of $T_{\phi,R}$ and adds to the domain of $\zeta$ those vertices representing terms in $\text{Name} \setminus \bar{n}$ (lines 7–8).

Now, suppose that $t = f(t_1, \ldots, t_n)$ for some $f \in \Sigma_n$ and some $t_1, \ldots, t_n \in T_\phi$. By the induction hypothesis, $(t_1)\downarrow, \ldots, (t_n)\downarrow \in \zeta_s[T_\phi]$. We consider two cases: either $f((t_1)\downarrow, \ldots, (t_n)\downarrow)$ is in normal form or not. In the first case, clearly (A) holds. To prove (B), note that if $f((t_1)\downarrow, \ldots, (t_n)\downarrow) = f((t_1)\downarrow, \ldots, (t_n)\downarrow)$ is represented by some vertex $v$ in $T_{\phi,R}$, then the same holds for $(t_1)\downarrow, \ldots, (t_n)\downarrow$. Let $v_1, \ldots, v_n$ be these vertices. Let $\text{dom}_k(\zeta)$ and $\text{dom}_k(\zeta_s)$ be the domains of $\zeta$ and $\zeta_s$ (respectively) after the $k$-th execution of the loop in lines 5–20. Let $\text{visit}_n$ denote the set $\text{visit}_n$ at the beginning of the $k$-th execution of this loop. Now, there is $k$ such that $\{v_i \mid i \in \{1, \ldots, n\}\} \in \text{dom}_{k+1}(\zeta) \setminus \text{dom}_k(\zeta)$; that is, at the $k$-th iteration of the loop in lines 5–20 all vertices $v_i$ have been added to $\text{dom}_k(\zeta)$. It is then clear that $v \in \text{visit}_n$. By the induction hypothesis, there are $\phi_s$-recipes $t'_1, \ldots, t'_n$ for $t_1\downarrow, \ldots, t_n\downarrow$ (respectively) such that $\text{term}_{T_{\phi,R}}(v_i) = t'_i\zeta_s$ for all $i$. Line 12 then ensures that $v \in \text{dom}(\zeta)$, and $\zeta(v)$ is a DAG-representation of $f(\text{term}_{T_{\phi,R}}(v_1), \ldots, \text{term}_{T_{\phi,R}}(v_n))$. This is the same as $f(t'_1, \ldots, t'_n)\zeta_s$, and $f(t'_1, \ldots, t'_n)$ is a $\phi_s$-recipe for $f((t_1)\downarrow, \ldots, (t_n)\downarrow) = (t\sigma)\downarrow$.

If $f((t_1)\downarrow, \ldots, (t_n)\downarrow)$ is not in normal form, then there is some rule $(l \to r) \in R$ and some substitution $\sigma_l : \text{vars}(l) \to T(\Sigma, \text{Name})$ such that $f((t_1)\downarrow, \ldots, (t_n)\downarrow) = l\sigma_l$. Since $r\sigma_l$ is a proper subterm of $l\sigma_l$, we conclude that $r\sigma_l$ is in normal form.

We again distinguish two cases, depending on whether or not $r\sigma_l \in \text{sub}(\text{ran}(\sigma))$. If $r\sigma_l \notin \text{sub}(\text{ran}(\sigma))$, then either $r\sigma_l \in T(\Sigma, \emptyset)$, or $r\sigma_l \in \text{sub}(l\sigma_l)$. In the first case (A) is clear. (B) holds because we have already proven that it holds for all the leaves of $T_{\phi,R}$.

Now, we consider the second case. We have $l\sigma_l \in \sigma_s[T_\phi]$, and it is easy to see that $\text{sub}(\text{ran}(\zeta_s)) \subseteq \text{sub}(\text{ran}(\sigma))$ (since all terms in the range of $\zeta_s$ are in $\text{sub}(\text{ran}(\sigma)) \cup T$, due to the check in line 15). Thus, by Lemma 8, $r\sigma_l \in \sigma_s[T_\phi]$, and again (A) holds.

Consider now the case that $r\sigma_l \in \text{sub}(\text{ran}(\sigma))$. For each $x \in \text{vars}(l)$, we have $x\sigma_l \in \text{sub}(f((t_1)\downarrow, \ldots, (t_n)\downarrow))$. By Lemma 8 and the fact that $f((t_1)\downarrow, \ldots, (t_n)\downarrow) \in \zeta_s[T_\phi]$, either $x\sigma_l \in \text{sub}(\text{ran}(\sigma))$ or $x\sigma_l \in \zeta_s[T_\phi]$. Take the
substitution \( \sigma'_i \); \( \text{vars}(l) \rightarrow T(\Sigma, \text{Name}) \) such that, for each \( x \in \text{vars}(l) \):

\[
x_{\sigma'_i} = \begin{cases} 
  x_{\sigma} & \text{if } x_{\sigma} \in \text{sub}(\text{ran}(\sigma)) \\
  \tau & \text{if } x_{\sigma} \notin \text{sub}(\text{ran}(\sigma)) 
\end{cases}
\]

Let \( l = f(l_1, \ldots, l_n) \). By Lemma 9, \( l_i \sigma'_i \in \varsigma_s[T_{\phi_s}] \) for each \( i \in \{1, \ldots, n\} \). Also, each \( l_i \) is in normal form, since it can be obtained by replacing subterms of \((t, \sigma)\) by \( \tau \). Moreover, as \( r \sigma_i \in \text{sub}(\text{ran}(\sigma)) \), we have that, for each \( x \in \text{vars}(v) \), \( x_{\sigma_i} \in \text{sub}(\text{ran}(\sigma)) \); thus, \( r \sigma_i = r \sigma'_i \). By Property (8) of Lemma 2, there is \( v \in V_{\phi,r} \) such that \( \text{term}_{\tau_s,n}(v) = l \sigma_i \), \( v \in \text{dom}(rw) \), and \( \text{term}_{\tau_s,n}(rw(v)) = r \sigma'_i \). Thus, there are vertices \( v_i \in V_{\phi,r} \) such that \( \text{term}_{\tau_s,n}(v_i) = l \sigma_i \) for each \( i \in \{1, \ldots, n\} \), and by the induction hypothesis we conclude that \( v_i \in \text{dom}(\varsigma) \) for each \( i \).

Now, there is \( k \) such that \( \{ v_i \mid i \in \{1, \ldots, n\} \} \in \text{dom}_{k+1}(\varsigma) \setminus \text{dom}_k(\varsigma) \); that is, at the \( k \)-th iteration of the loop in lines 5–20 all vertices \( v_i \) have been added to \( \text{dom}(\varsigma) \). It is then clear that \( v \in \text{vis} \text{now}_k+1(\varsigma) \). By a reasoning similar to the one used above, line 12 then ensures that \( \varsigma(v) \) is a DAG-representation of \( t' \varsigma_s \), where \( t' \) is a fresh \( \phi \)-recipe for \( r \sigma'_i \). Now, if \( x \) is the chosen fresh variable, then \( x_{\sigma_s} = r \sigma'_i \), and line 17 ensures that \( r \sigma'_i = (t \sigma)_r \in \text{ran}(\sigma_s) \). Lines 14 and 18 ensure that \( \varsigma(rw(v)) = x_{\varsigma_s} \), and \( x \) is a fresh \( \phi \)-recipe for \( r \sigma'_i = \text{term}_{\tau_s,n}(rw(v)) \). The fact that \( x_{\varsigma_s} \in T_{\phi} \) follows from the induction hypothesis as before. This proves both (A) and (B).

\( \phi_s \) is a saturation due to the property (A) just proved and from the fact that, for all \( x \in \text{dom}(\sigma_s) \), \( \varsigma_s(x) \) is a DAG-representation of a \( \phi \)-recipe for \( \sigma_s \).

To see that \( \varsigma_s(v) \in O(|\phi|) \), we prove that in fact \( \varsigma(v) \in O(|\phi|) \) for all \( v \in \text{dom}(\varsigma) \cap V_{\phi} \). To prove this, suppose that a vertex \( v \) is added to the domain of \( \varsigma \). This can happen executing either line 12 or line 14. In line 12, it is clear that only one vertex is added to the vertices occurring in the DAGs \( \varsigma(v) \) for \( v \in \text{dom}(\varsigma) \cap V_{\phi} \). In line 14, \( \varsigma(rw(v)) \) is a DAG-representation of an instance of the left-hand side of a rewrite rule whose variables are instantiated with elements of \( \text{sub}(\text{ran}(\sigma)) \cup T \). Thus, \( x_{\varsigma_s} \) contains at most a constant number \( |l| \) of variables that do not occur in the DAGs \( \varsigma(v) \) for \( v \in \text{dom}(\varsigma) \cap V_{\phi} \). We conclude that the set of vertices occurring in the DAG-representations \( \varsigma(v) \) for \( v \in \text{dom}(\varsigma) \cap V_{\phi} \) is augmented at most \( |\text{sub}(\text{ran}(\sigma))| \) times, each time adding at most \( O(|l|) \) vertices. Thus, the set of vertices occurring in a DAG \( \varsigma(v) \) for some \( v \in \text{dom}(\varsigma) \cap V_{\phi} \) has at most \( O(|\phi|) \) elements. Since, whenever we increase the range of \( \varsigma_s \) with a new element \( x_{\varsigma_s} \) (in line 18), \( x_{\varsigma_s} = \varsigma(rw(v)) \) for some vertex \( rw(v) \in \text{dom}(\varsigma) \cap V_{\phi} \), it follows that \( |x_{\varsigma_s}| \in O(|\phi|) \).

To prove that \( \text{term}_{\tau_s,n}(v) \in \sigma_s[T_{\phi_s}] \) implies \( v \in \text{dom}(\varsigma) \) we proceed by induction. Observe first that all vertices representing terms in \( \text{ran}(\sigma) \) are in the domain of \( \varsigma \), due to property (6) of 2. All other terms added to the range of \( \sigma_s \) are added in the execution of line 16. In this case, it is clear that \( rw(v) \) is the only vertex in \( T_{\phi,R} \) representing the new element \( x_{\sigma_s} \) in the range of \( \sigma_s \), and \( rw(v) \in \text{dom}(\varsigma) \), so that the property still holds. If \( \text{term}_{\tau_s,n}(v) \in \text{Name} \setminus \bar{n} \), then \( v \in \text{leaves}(T) \), and the first execution of the loop in lines 5–20 ensures that \( v \in \text{dom}(\varsigma) \). Finally, if \( \text{term}_{\tau_s,n}(v) = f(t_1, \ldots, t_n) \) for some \( t_1, \ldots, t_n \in \sigma_s[T_{\phi_s}] \), then there are vertices \( v_1, \ldots, v_n \) such that \( \text{term}_{\tau_s,n}(v_i) = t_i \) for all \( i \), and, by
the induction hypothesis, $v_i \in \text{dom}(\zeta)$ for all $i$. By a reasoning similar to the one used above, there is a minimum $k$ such that $v_1, \ldots, v_n \in \text{dom}_k(\zeta)$, and it follows that $v \in \text{dom}_k(\zeta) \subseteq \text{dom}(\zeta)$. This concludes the proof.

To estimate the complexity of the algorithm, observe that the number of vertices in $\mathcal{T}_{\phi,R}$ is in $O(|\phi|^\text{vars}(R))$. Each leaf is visited once, and each other vertex is visited at most once for each outgoing edge (because a vertex $v$ is added to the list of vertices to visit in the next execution of the loop if, in the current execution of the loop, the algorithm adds to the domain of $\zeta$ a vertex $v'$ such that there is an edge from $v$ to $v'$). We obtain that the total number of visits to vertices is $O(|\phi|^\text{vars}(R))$.

Consider now each visit to a vertex. Unless the condition on line 15 is verified, all operations performed by the visit take time at most $O(\log |\phi|)$. If this condition is verified, the visit takes time $O(|\phi| \log |\phi|)$, since we need to assign to $x\sigma$ a term with size $O(|\phi|)$. The condition on line 15 is only verified at most $O(|\phi|)$ times, since it adds to the domain of $\zeta$ a vertex representing a term in $\text{sub}(\text{ran}(\sigma)) \cup \{\tau\}$ and there are only $O(|\phi|)$ such terms. Note that to efficiently decide whether the condition on line 15 holds we must store the information of whether the term represented by a vertex is a subterm of the range of $\sigma$ locally, e.g., as a boolean field in the object representing that vertex. Combining this complexity with the complexity of constructing the forest $\mathcal{T}_{\phi,R}$, we obtain a total complexity of at most $O(|\phi|^\text{vars}(R) \log^2 |\phi|)$.

\section{Section 3.2}

\textbf{Proof (Theorem 1).} By Lemma 5, computing (a minimal DAG-representation of) the normal form $t \downarrow$ of $t$ takes time $O(|t| \log^2 |t|)$. By Lemma 3, computing the saturated frame $\phi_s$ takes time $O(|\phi|^\text{vars}(R) \log^2 |\phi|)$. By Lemma 6, deciding whether $t \downarrow \in \sigma_s[\mathcal{T}_{\phi_s}]$ can be done in time $O(|\phi_s| |t|)$ - that is, $O(|\phi|^2 |t|)$, since $|\phi_s| \in O(|\phi|^2)$. We thus obtain an overall complexity of $O(|t| \log^2 |t| + |\phi|^2 |t| + |\phi|^\text{vars}(R) \log^2 |\phi|)$ for the whole procedure. \hfill \qed

\section{Section 3.3}

The next lemma states a well-known property of equational theories generated by rewriting systems: equality is stable under substitution of names for terms. As no names occur in rewrite rules, one can apply the same rewriting sequence after replacing the names by terms.

\textbf{Lemma 10.} Let $R$ be a convergent rewriting system, $t, t' \in T(\Sigma, \text{Name} \cup \text{Var})$, $\sigma: \text{vars}(t) \cup \text{vars}(t') \rightarrow \text{Name}$ be an injective substitution such that $\text{ran}(\sigma) \cap \text{sub}(t) = \text{ran}(\sigma) \cap \text{sub}(t') = \emptyset$, and $\sigma': \text{vars}(t) \cup \text{vars}(t') \rightarrow T(\Sigma, \text{Name})$ be any substitution. Then $\sigma \approx_R t' \sigma$ implies $\sigma' \approx_R t' \sigma'$.

\footnote{Note, however, that we do not need to store $\zeta(\text{rw}(v))$ in $x\zeta$; it is enough to store this DAG-forest and the functions $\zeta$ and $\text{rw}$, and store at $x\zeta$ the name of the vertex $v$. The DAG-forest $\zeta(\text{rw}(v))$ can then be retrieved in $O(|\phi| \log |\phi|)$.}
Lemma 11. Let \( R \) be a subterm convergent rewriting system and suppose that the frames \( \phi \) and \( \phi' \) are such that \( T_{\phi'} = T_{\phi} \). Assume that, for all \( x \in \text{dom}(\sigma) \) and \( t' \in T_{\phi'} \), \( x\sigma = t'\sigma \Rightarrow x\sigma' \approx_R t'\sigma' \). Then, for all \( t,t' \in T_{\phi} \), we have \( t\sigma = t'\sigma \Rightarrow t\sigma' \approx_R t'\sigma' \).

Proof. The lemma is easily proved by induction on \( t \). For the base case, we must prove that the result holds for \( t \in \text{Name} \setminus \bar{n} \) and for \( t \in \text{dom}(\sigma) \). If \( t \in \text{dom}(\sigma) \), the result is trivial. If \( t \in \text{Name} \setminus \bar{n} \), then \( t\sigma = t\sigma \) implies \( t\sigma' = t\sigma' \). Thus, \( t' \in \text{dom}(\sigma) \) or \( t' = t \). In both cases, the result follows.

Now let \( t = f(t_1, \ldots, t_n) \) for some \( t_1, \ldots, t_n \in T_{\phi} \) and some \( f \in \Sigma_n \). By the induction hypothesis, \( t_1\sigma = t_1\sigma \) implies \( t_1\sigma' \approx_R t_1'\sigma' \). Now, \( t\sigma = t\sigma \) implies that either \( t' \in \text{dom}(\sigma) \) or \( t' = f(t_1', \ldots, t_n') \) for some \( t_1', \ldots, t_n' \in T_{\phi} \) such that \( t_i\sigma = t_i\sigma \) for all \( i \in \{1, \ldots, n\} \). In the first case, the result is trivial. In the second, it follows from the induction hypothesis.

Lemma 12. Let \( R \) be a subterm convergent rewriting system. Assume that for all rules \((l \rightarrow r) \in R \), all substitutions \( \sigma; \text{vars}(l) \rightarrow \text{sub}(\text{ran}(\sigma)) \cup T \), all and recipes \( t_i, t_r \in \sigma[T_{\phi}] \) such that \( l\sigma_i = r\sigma_i \) and \( r\sigma_i = r\sigma_i \), we have \( t_i\sigma' \approx_R t_i'\sigma' \). Then, for all \( t \in T_{\phi} \), there is a \( t_{\text{nf}} \in T_{\phi} \) such that \( (t\sigma)_\downarrow = t_{\text{nf}}\sigma \) and \( t\sigma \approx_R t_{\text{nf}}\sigma' \).

Proof. The proof is by induction on \( t \). In the base case, consider \( t \in \text{Name} \setminus \bar{n} \) and \( t \in \text{dom}(\sigma) \). For both cases we may choose \( t_{\text{nf}} = t \), and the result follows.

Now suppose that \( t = f(t_1, \ldots, t_n) \) for some \( t_i \in \Sigma_n \) and some \( t_1, \ldots, t_n \in T_{\phi} \). By the induction hypothesis, there are \( t_{1,\text{nf}}, \ldots, t_{n,\text{nf}} \in T_{\phi} \) such that \( t_{i,\text{nf}}\sigma = (t_i\sigma)_\downarrow \) and \( t_i\sigma \approx_R t_{i,\text{nf}}\sigma' \) for all \( i \in \{1, \ldots, n\} \).

If \( f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}})\sigma_s \) is in normal form, we simply take \( t_{\text{nf}} = f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}}) \). Suppose then that \( f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}})\sigma_s \) is not in normal form. Then there is some rule \((l \rightarrow r) \in R \) and some substitution \( \sigma; \text{vars}(l) \rightarrow T(\Sigma, \text{Name}) \) such that \( l\sigma_i = f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}})\sigma_s \). By Lemma 7, we conclude that \( r\sigma_i \) is in normal form. By the definition of saturation, there is \( t_r \in T_{\phi} \) such that \( t_r\sigma = r\sigma_i \). We now need to prove that \( f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}})\sigma_s \approx_R t_r\sigma' \).

The technique is similar to the one used in the proof of decidability of deduction. First, we determine an injective function \( \rho; \text{vars}(l) \rightarrow T \) (we can do this because, by construction, \( T \) is at least as large as the set of variables occurring in any rule). We define the substitution \( \sigma'; \text{vars}(l) \rightarrow \text{sub}(\text{ran}(\sigma)) \cup T \) by:

\[
x\sigma' = \begin{cases} 
x\sigma \text{ if } x\sigma \in \text{sub}(\text{ran}(\sigma)) 
x\rho(x) \text{ if } x\sigma \notin \text{sub}(\text{ran}(\sigma)).
\end{cases}
\]

By Lemma 9, \( l\sigma_i, r\sigma_i \in \sigma[T_{\phi}] \). Thus, there are \( t_i', t_r' \in T_{\phi} \) such that \( l\sigma_i = t_i'\sigma_s \) and \( r\sigma_i = t_r'\sigma_s \). Now, \( t_i\sigma' \approx_R t_i'\sigma' \) by hypothesis, and Lemma 10 implies \( t_r = f(t_{1,\text{nf}}, \ldots, t_{n,\text{nf}})\sigma_s \approx_R t_r\sigma' \) as desired.

Lemma 13. Let \( R \) be a subterm convergent rewriting system. There are \( t, t' \in T_{\phi} \) with \( \sigma \approx_R t\sigma \) and \( t\sigma \not\approx_R t'\sigma' \) if and only if one of the following conditions holds:
(1) there are \( x \in \text{dom}(\sigma_s) \) and \( u \in T_{\phi_s} \), such that \( x\sigma_s = u\sigma_s \) and \( x\zeta\sigma' \not\approx_R u\zeta\sigma' \);
(2) there is a rule \((l \rightarrow r) \in R \) and a substitution \( \sigma_l : \text{vars}(l) \rightarrow \text{sub(\text{ran}(\sigma_l))} \cup \mathcal{T} \)

such that, for some \( t_l, t_r \in T_{\phi_s}, l\sigma_l = t_l\sigma_s, r\sigma_l = t_r\sigma_s, \) and \( l\zeta\sigma' \not\approx_R t_r\zeta\sigma'. \)

Proof. (\( \Rightarrow \)) If (1) holds, one may choose \( t = x \) and \( t' = u\zeta; \) if (2) holds, one may choose \( t = t_l\zeta \) and \( t' = t_r\zeta. \)

(\( \Leftarrow \)) Let \( t, t' \in T_{\phi_s} \) be such that \( \sigma \approx_R t'\sigma. \) We show that if neither condition holds then \( \sigma \approx_R t'\sigma. \) Since \( \sigma \approx_R t'\sigma, \) we have \( (\sigma)\downarrow = (t'\sigma)\downarrow \) and we know by Lemma 1 that \((\sigma)\downarrow \in \sigma_s[T_{\phi_s}]. \) By Lemma 12, there are terms \( t_{df}, t'_{df} \in T_{\phi_s} \) such that \( t_{df}\sigma_s = (\sigma)\downarrow \) and \( t'_{df} \approx_R t_{df}\zeta\sigma', \) and analogously for \( t'_{df} \) and \( t'. \) Note that the negation of (2) is precisely the premise of Lemma 12. Condition (1) now yields the result, as \( t' \zeta\sigma' \approx_R t'_{df}\zeta\sigma' \approx_R t'\sigma', \) where we use Lemma 11 and the fact that \( t_{df}\sigma_s = (\sigma)\downarrow = (t'\sigma)\downarrow = t'_{df}\sigma_s. \)

Proof (Theorem 2). We prove that lines 1-11 output \text{false} if and only if there are \( t, t' \in T_{\phi_s} \) such that \( \sigma \approx_R t'\sigma \) and \( t' \not\approx_R t'\sigma'. \) Then line 12 outputs \text{false} if and only if the same property holds exchanging \( \phi \) and \( \phi'. \) If neither returns \text{false}, then \( \phi \approx_R \phi', \) and the algorithm returns \text{true} as desired.

It is clear that if the cycle in lines 5-8 returns \text{false} then \( x_{i(k)}\zeta_s \) and \( t\zeta_s \) are \( \phi \)-recipes such that \( x_{i(k)}\zeta\sigma \approx_R t\zeta\sigma \) and \( x_{i(k)}\zeta\sigma' \not\approx_R t\zeta\sigma'. \) Thus, the two frames are not statically equivalent, and the algorithm outputs the correct result. Similarly, if the cycle in lines 9-11 returns \text{false}, then \( \zeta(v) = \zeta(\omega(v)) \) are two \( \phi \)-recipes that prove that the two frames are not statically equivalent, and the algorithm correctly outputs \text{false}.

To prove the converse, we show that: (A) condition (1) in Lemma 13 implies that the loop in lines 5-8 returns \text{false}; and (B) condition (2) in Lemma 13 implies that there is some vertex \( v \) for which the condition on lines 10 does not hold, and the cycle in lines 9-11 returns \text{false}.

We show that if condition (1) of Lemma 13 holds, then there is a \( k \in \{1, \ldots, m_s\} \) such that \( x_{i(k)}\sigma_s = t\sigma_s \) for some \( t \in \phi_{s,k-1}. \) Suppose then that there is \( x \in \text{dom}(\sigma_s) \) and \( t \in T_{\phi_s} \) such that \( x\sigma_s = t\sigma_s \) and \( x\zeta\sigma' \not\approx_R t\zeta\sigma'. \) Let \( k \) be such that \( x = x_{i(k)}. \) There are two cases, depending on whether or not \( t \in \text{dom}(\sigma_s). \)

If \( t \in \text{dom}(\sigma_s), \) then there is \( k' \) such that \( t = x_{i(k')}. \) Thus, either \( k' > k \) or \( k > k' \) (if \( k = k' \) then \( x\zeta\sigma' \approx_R t\zeta\sigma' \)). If \( k' > k \), then \( x_{i(k')}\sigma_s = x_{i(k')}\sigma_s \) and \( x_{i(k')} \in T_{\phi_{(k')}}. \) We reason analogously if \( k > k'. \)

If \( t \not\in \text{dom}(\sigma_s), \) let \( x \in \text{vars}(t). \) We have \(|t\sigma_s| > |x\sigma_s| \) for all \( x \in \text{vars}(t) \) and \( x_{i(k)}\sigma_s = t\sigma_s. \) Hence, \(|x_{i(k)}\sigma_s| > |x\sigma_s|, \) and \( t \in T_{\phi_{s,k-1}} \) since \( x_{i(k)} \not\in \text{vars}(t) \) for all \( k' \geq k. \)

Now we only need to prove that it does not matter which recipe we choose in line 6. Suppose that the iterator \( i \) in this loop reaches \( k. \) Then, for all \( j \in \{0, \ldots, k-1\}, \) there is \( t \in T_{\phi_{s,j+1}} \) such that \( x_{i(j)}\sigma_s = t\sigma_s \) and \( x_{i(j)}\zeta\sigma' \not\approx_R t\zeta\sigma'. \) From what we have just proved, it follows that there is no \( x \in \{x_{i(1)}, \ldots, x_{i(k-1)}\} \) and \( t \in T_{\phi_{s,k-1}} \) such that \( x\sigma_s = t\sigma_s \) and \( x\zeta\sigma' \not\approx_R t\zeta\sigma'. \) From Lemma 11, we conclude that for all \( t, t' \in T_{\phi_{s,k-1}}, t\sigma_s = t'\sigma_s \) implies \( t\zeta\sigma' \approx_R t'\zeta\sigma'. \) Thus, if \( t \) and \( t' \) are \( \phi_{s,k-1} \)-recipes for \( x_{i(k)} \), we have \( t\zeta\sigma' \approx_R t'\zeta\sigma, \) and it does not matter which we choose.
To prove (B), suppose that condition (2) in Lemma 13 holds for a rewrite rule \( l \rightarrow r \), a substitution \( \sigma \) and \( \phi \)-recipes \( t_l \) and \( t_r \) for \( l \sigma_1 \) and \( r \sigma_1 \), respectively. Then, by property (8) of Lemma 2, there is a vertex \( v \in \text{dom}(rw) \) such that \( \text{term}_{T_{\phi}}(v) = l \sigma_1 \). Since \( l \sigma_1 \in \sigma_1[T_{\phi},] \), we conclude, by Lemma 3, that \( v \in \text{dom}(\zeta) \), and \( \zeta(v) \) is a (DAG-representation of) a \( \phi \)-recipe for \( l \sigma_1 \). Similarly, \( rw(v) \in \text{dom}(\zeta) \), and \( \zeta(rw(v)) \) is a (DAG-representation of) \( r \sigma_1 \). We have \( \zeta(v) \sigma = t_l \sigma = l \sigma_1 \). By Lemma 11, either condition (1) of Lemma 13 holds, and the correct output false has already been output by the first loop, or we have \( \zeta(v) \sigma' \approx_R t_l \sigma' \). Similarly, we conclude that either the first loop has already returned false or \( \zeta(rw(v)) \sigma' \approx_R t_r \sigma' \). Thus, if \( \zeta(v) \sigma' \approx_R (\zeta(rw(v))) \sigma' \), we conclude that \( t_l \sigma' \approx_R t_r \sigma' \), which contradicts our assumption. It follows that \( (\zeta(v)) \sigma' \neq_R (\zeta(rw(v))) \sigma' \), and the algorithm outputs the correct result false.

Choosing the bijection and storing the \( \phi_{i,k,l} \) on lines 2 and 3 takes time \( O(|\phi|^3 \log |\phi|) \). Deciding whether \( x_{i(k)} \sigma \in \sigma_1[T_{\phi_{i,k,l}}] \) can be done in time \( |\phi|^3 \), and the comparison on line 8 takes time \( |\phi| \log^2 |\phi| \). The loop in lines 9–11 requires comparing \( O(|\phi|^{\text{vars}(R)}) \) pairs of recipes with size \( O(|\phi| + |\phi'|) \) for equality, with total complexity \( O((|\phi| + |\phi'|)^{\text{vars}(R)} + 1 \log^2(|\phi| + |\phi'|)) \). Since this operation will also be performed exchanging the roles of \( \phi \) and \( \phi' \) (as part of the execution of line 12), Lemma 5 yields a time complexity of at most \( O((|\phi| + |\phi'|)^3 + (|\phi| + |\phi'|)^{\text{vars}(R)} + 1 \log^2(|\phi| + |\phi'|)) \).

Section 4

The following lemma is analogous to Lemma 7 and will be useful for proving the correctness of our algorithms for the prefix theory.

**Lemma 14.** Let \((l \rightarrow r) \in R_p \) and \( t = f(t_1, \ldots, t_n) \) be such that all \( t_i \) are in normal form and \( t = l \sigma_1 \) for some \( \sigma : \text{vars}(l) \rightarrow T(\Sigma, \text{Name}) \). Then \( r \sigma_1 \) is in normal form.

**Proof.** For all rules \((l \rightarrow r) \in R_p \) except \((\pi_1(\{(x, y)\}_2^t) \rightarrow \{x\}_2^t) \) we have \( l \in \text{sub}(r) \). In such cases, the proof of Lemma 7 suffices. For the prefix rule, we simply note that \( t = l \sigma_1 \) then implies \( t = \{(t_1, t_2)\}_{t_3}^t \), for some \( t_1, t_2, t_3 \in T(\Sigma, \text{Name}) \) in normal form. Since there is no rule whose left-hand side is of the form \( \{t_1\}_{t_2}^t \), we conclude that \( r \sigma_1 = \{t_1\}_{t_2}^t \) is in normal form.

**Lemma 15.** \( |\text{sub}_p(t)| \leq 2|t| \) for all \( t \in T(\Sigma^D, \text{Name}) \), and \( |\text{sub}_p(\text{ran}(\sigma))| \in O(|\phi|) \).

**Proof.** We prove the result by induction. The base case is \( t \in \text{Name} \). In this case, \( |\text{sub}_p(t)| = |t| = 1 \) and the result holds.

Now suppose that \( t = f(t_1, \ldots, t_n) \). There are two cases:

(A) \( t = \{(t_1', t_2')\}_{t_3}^t \), for some \( t_1', t_2', t_3' \in T(\Sigma^D, \text{Name}) \),

(B) (A) does not hold.
We consider first case \((B)\). Then, \(\text{sub}_p(t) = \{t\} \cup \bigcup_{i=1}^{n} \text{sub}_p(t_i)\). By the induction hypothesis, we conclude that
\[
|\text{sub}_p(t)| \leq 1 + \sum_{i=1}^{n} |\text{sub}_p(t_i)| \leq 2 + 2 \sum_{i=1}^{n} |t_i| = 2|t|.
\]

We now prove the result for case \((A)\). If \(t = \{\langle t'_1, t'_2 \rangle\}_{t_3}^{s}\), there is some \(m \geq 2\) and terms \(p_1, \ldots, p_m\) such that \(t = \{\langle p_1, \ldots, p_m \rangle\}_{t_3}^{s}\) and \(p_1 \neq \langle p'_1, p'_2 \rangle\) for all \(p'_1, p'_2 \in T(\Sigma^{DY}, \text{Name})\) (note that \(p_m = t_3\)). For each \(i \in \{1, \ldots, m\}\), let \(t_i = \{\langle p_1, \ldots, p_i \rangle\}_{t_3}^{s}\), so that \(t_m = t\). By the definition of \(\text{sub}_p(t)\), we obtain
\[
\text{sub}_p(t) = \{t\} \cup \text{sub}_p(\langle p_1, \ldots, p_{m-1} \rangle) \\
\cup \text{sub}_p(p_m) \\
\cup \text{sub}_p(t_3) \\
\cup \text{sub}_p(t_{m-1}).
\]

Applying the same formula for \(t_{m-1}, \ldots, t_1\), we conclude that
\[
\text{sub}_p(t) = \{t_i \mid i \in \{1, \ldots, m\}\} \\
\cup \{\langle p_1, \ldots, p_i \rangle \mid i \in \{1, \ldots, m\}\} \\
\cup \bigcup_{i=1}^{m} \text{sub}_p(p_i) \\
\cup \text{sub}_p(t_3).
\]

Thus,
\[
|\text{sub}_p(t)| \leq 2m + |\text{sub}_p(t_3)| + \sum_{i=1}^{m} |\text{sub}_p(p_i)|;
\]
and, by induction hypothesis,
\[
|\text{sub}_p(t)| \leq 2m + 2|t_3| + 2 \sum_{i=1}^{m} |p_i|.
\]

In contrast, we obtain the following expression for \(|t|\):
\[
|t| = 1 + |\langle p_1, \ldots, p_{m-1}, p_m \rangle| + |t_3| \\
= 1 + (1 + |\langle p_1, \ldots, p_{m-1} \rangle| + |p_m|) + |t_3| \\
= \ldots \\
= 1 + (m - 1) + |t_3| + \sum_{i=1}^{m} |p_i| \\
= m + |t_3| + \sum_{i=1}^{m} |p_i|.
\]

Combining the two results above we conclude that \(|\text{sub}_p(t)| \leq 2|t|\). The second statement is now clear:
\[
|\text{sub}_p(\text{ran}(\sigma))| \leq \sum_{t \in \text{ran}(\sigma)} |\text{sub}_p(t)| \leq 2 \sum_{t \in \text{ran}(\sigma)} |t| \leq 2|\varphi|.
\]
\(\square\)
Algorithm 5 (Computing $T_\phi$)
Input: a frame $\phi = v_1.\sigma$, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
Output: a DAG-forest $T_\phi$ and the functions $\zeta$, $r_\phi$

1: let $T_1, \ldots, T_{n+3}$ be DAG representations of $t_1, \ldots, t_{n+3}, T_1, T_2, T_3$ with roots $v_1, \ldots, v_{n+3}$ and disjoint sets of vertices
2: let $T_\phi = (V_\phi, \lambda_\phi, E_\phi)$ and $\min$ be the output of Algorithm 3, on input $\bigcup_{i=1}^{n+3} T_i$
3: $T_{\phi, p} = (V_{\phi, p}, l_{\phi, p}, E_{\phi, p}) \leftarrow T_\phi$
4: $\zeta \leftarrow \{\min(v_{n+i}) \mapsto \{(v_{n+i}, \{v_{n+i} \mapsto t_i\}, \emptyset) \mid i \in \{1, 2, 3\}\}$
5: for $i = 1$ to $n$
6: if $\min(v_i) \notin \text{dom}(\zeta)$
7: then $\zeta \leftarrow (\zeta \cup \{v \mapsto ((v_i, \{v_i \mapsto x_i\}, \emptyset))\})$
8: $r_\phi = \emptyset$
9: $\text{for all } (l \mapsto r) \in R_p$
10: $\text{for all } \sigma; \text{vars}(l) \to V_\phi$
11: $\text{for all } (l \mapsto r) \in R_{\phi, p}, \sigma; \text{vars}(l) \to \text{sub}_p(\text{ran}(\sigma)) \cup \emptyset$
12: $\text{add to } T_{\phi, p} \text{ vertices to represent } l_{\sigma_1} \text{ and } r_{\sigma_1}, \text{ keeping } T_{\phi, p} \text{ minimal}$
13: $r_\phi \leftarrow r_\phi \cup \{v_l \mapsto v_r\}$, where term$\phi_{\tau_{\sigma_1}}(v_l) = l_{\sigma_1}$ and term$\phi_{\tau_{\sigma_1}}(v_r) = r_{\sigma_1}$

Lemma 16. The forest $T_{\phi, p}$ and the functions $r_\phi$ and $\zeta$ are such that:

1. $T_{\phi, p}$ can be obtained in time $O(|\phi|^3 \log^2 |\phi|)$, and $|T_{\phi, p}| \in O(|\phi|^3)$;
2. $r_\phi$ can be computed in time $O(|\phi|)$;
3. $\zeta$ can be computed in time $O(|\phi| \log |\phi|)$ and for each $v \in \text{dom}(\zeta)$, $|\zeta(v)| \in O(1)$;
4. for each $s \in \text{sub}_p(\text{ran}(\sigma)) \cup \emptyset$, there is one and only one $v \in V_{\phi, p}$ such that term$\phi_{\tau_{\sigma_1}}(v) = s$;
5. for each $v \in \text{dom}(r_\phi)$, term$\phi_{\tau_{\sigma_1}}(v) \to_R \text{term} \phi_{\tau_{\sigma_1}}(r_\phi)$;
6. for each $t \in \text{ran}(\sigma) \cup \emptyset$, there is $v$ such that term$\phi_{\tau_{\sigma_1}}(v) = t$ and $v \in \text{dom}(\zeta)$;
7. if $v \in \text{dom}(\zeta)$, then $\zeta(v)$ is a DAG-representation of a $\phi$-recipe for term$\phi_{\tau_{\sigma_1}}(v)$;
8. if $(l \mapsto r) \in R_p$ is a rewrite rule and $\sigma_1; \text{vars}(l) \to \text{sub}_p(\text{ran}(\sigma)) \cup \emptyset$ is a substitution, there is $v \in V_{\phi, p} \cap \text{dom}(\text{term} \phi_{\tau_{\sigma_1}}(v))$ such that term$\phi_{\tau_{\sigma_1}}(v) = l_{\sigma_1}$ and term$\phi_{\tau_{\sigma_1}}(r_\phi(v)) = r_{\sigma_1}$.

Proof. The proof is analogous to the proof of Lemma 2. We note only two small adaptation. First, our complexity analysis rely on the fact that, by Lemma 15, the forest $T_\phi$ defined in line 2 has size $|T_\phi| \in O(|\phi|)$. Second, Algorithm 3 can be trivially adapted to work for the prefix theory without increasing its complexity: it is enough to ensure that, in line 11, we create the vertex $v_e$ representing the instance $r_{\sigma_1}$ of the right-hand side of a rule. This involves adding at most a constant number of vertices to $T_{\phi, p}$, and can be done in $O(|\log^2 |\phi|)$ time, including the checks to ensure that the tree is still minimal. In light of Lemma 14, the term represented by this vertex is in normal form.

Proof (Lemma 4). The proof of this Lemma follows closely the proof of Lemma 3, replacing the natural instances of $\text{sub}(\phi)$ by $\text{sub}_p(\phi)$. We only need to adapt
our argument for proving that \( (t\sigma) \in \sigma_n[T_{\phi}] \) for all \( t \in T_{\phi} \). This proof is by induction, and the base case can be proved as in Lemma 3. For the step, we now need to consider the extra case that \( t_1, \ldots, t_n \in T_{\phi} \) and \( f((t_1\sigma) \downarrow, \ldots, (t_n\sigma)) \in T_{\phi} \) is an instance of the rewrite rule \( \pi_{\sigma}(\{x,y\}_z^\phi) \rightarrow \{x\}_z^\phi \) (note that, for all other rules \( (l \rightarrow r) \in R_p \), we have \( r \in sub(l) \), so the proof used in Lemma 3 is still valid). By the induction hypothesis, there are \( u_1, \ldots, u_n \) such that \( u_i \in T_{\phi} \) and \( u_i\sigma_i = (t_i\sigma) \downarrow \) for all \( i \in \{1, \ldots, n\} \), and we have \( f(u_1, \ldots, u_n)\sigma_s = \pi_{\sigma}(\{(t_1,t_2)_2^\phi\}) \) for some \( t_1, t_2, t_3 \in T(\Sigma, \text{Name}) \) in normal form.

By Lemma 14, \( r\sigma_l \) is in normal form. Since \( t \) is not in normal form, it follows that \( t \notin sub(\text{ran}(\sigma)) \). Moreover, since \( t \) is not of the form \( \{t_1\}^*_{t_2} \) then \( t \notin sub_p(\text{ran}(\sigma)) \). Since \( \text{ran}(\sigma_s) \subseteq \text{sub}_p(\text{ran}(\sigma)), \) it follows that \( t \notin \text{ran}(\sigma_s) \). Thus, since \( t \in \sigma_s[T_{\phi}] \), we must have \( \{t_1,t_2\}^{t_3} \in \sigma_s[T_{\phi}] \).

There are two cases: (A) \( \{t_1,t_2\}, t_3 \in \sigma_s[T_{\phi}] \) and (B) \( \{t_1,t_2\}^{t_3} \in \text{ran}(\sigma_s) \).

For case (A), we have two subcases: (A.1) \( t_1, t_2 \in \sigma_s[T_{\phi}] \) and (A.2) \( \{t_1,t_2\} \in \text{ran}(\sigma_s) \). For (A.1), we have \( t_1, t_2, t_3 \in \sigma_s[T_{\phi}] \). Thus, \( \{t_1\}^{t_3} \in \sigma_s[T_{\phi}] \) as desired. Consider now (A.2). In this case, we observe that the vertex \( v_1 \) representing \( \pi_{\sigma}(\{(t_1,t_2)_2^\phi\}) \) will be added to the range of \( \zeta \), since it is of the form \( \ell_\zeta \) for the rewrite rules \( \pi_1(\{x,y\}_z^\phi) \rightarrow x \) and the substitution \( \sigma_1 = \{x \mapsto t_1, y \mapsto t_2\} \) whose range is contained in \( \text{sub}_p(\text{ran}(\sigma)) \subseteq T \). Then, line 17 ensures that \( \text{term}_{T_{\phi}}(\text{rw}(v_1)) = t_1 \) and \( \text{term}_{T_{\phi}}(\text{rw}(v_2)) = t_2 \) are in the range of \( \sigma_s \). Since \( t_3 \in \sigma_s[T_{\phi}] \) (by definition of case (A)), we conclude that \( \{t_1\}^{t_3} \in \sigma_s[T_{\phi}] \).

In case (B), \( \{t_1,t_2\}^{t_3} \in \text{ran}(\sigma_s) \). It is clear that \( t_1, t_2, t_3 \in \text{sub}(\text{ran}(\sigma_s)) \). Since \( \text{ran}(\sigma_s) \subseteq \text{sub}_p(\text{ran}(\sigma)), \) we conclude that \( \{t_1\}^{t_3} \in \text{sub}_p(\text{ran}(\sigma)). \) By a reasoning similar to the one used above, we conclude that the vertex \( v \) representing \( \pi_1(\{x,y\}^\phi) \) will eventually be added to the range of \( \zeta \), and again line 17 ensures that \( \text{term}_{T_{\phi}}(v) = \{t_1\}^{t_3} \in \text{ran}(\sigma_s), \) which concludes the proof.

\textbf{Proof (Theorem 3).} The proof is analogous to the proof of Lemma 1, noting that \( \text{nvars}(R_p) = 3 \).

\textbf{Proof (Theorem 4).} The proof is analogous to the proof of Lemma 2, noting that \( \text{nvars}(R_p) = 3 \).

\textbf{Section 4.1}

In Tables 2 and 3, we let \( \sigma_w \) and \( \sigma_g \) be the substitutions such that \( v\tilde{u}\sigma_w \) and \( v\tilde{u}\sigma_g \) are saturations of \( \phi_w \) and \( \phi_g \), respectively. We adopt a similar convention for the substitutions \( \zeta_w \) and \( \zeta_g \) providing recipes for \( \sigma_w \) and \( \sigma_g \).
Table 2. Saturation of the Frame $\phi_w$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_1,\sigma_w$</th>
<th>$x_1,\zeta_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>${{T, K_{A,TGS}, t_1}}<em>{K</em>{A,KAS}}^{*}$</td>
<td>$x_0$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>${{T, K'<em>{A,TGS}, t_2}}</em>{K_{A,KAS}}^{*}$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>${{T, K_{A,TGS}, t_1}}<em>{K</em>{A,KAS}}^{*}$</td>
<td>$\pi_1(x_0)$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>${{T, K'<em>{A,TGS}, t_2}}</em>{K_{A,KAS}}^{*}$</td>
<td>$\pi_1(x_1)$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>${{T, K_{A,TGS}, t_1}}<em>{K</em>{A,KAS}}^{*}$</td>
<td>$\pi_1(\pi_1(x_0))$</td>
</tr>
<tr>
<td>$x_5$</td>
<td>${{T, K'<em>{A,TGS}, t_2}}</em>{K_{A,KAS}}^{*}$</td>
<td>$\pi_1(\pi_1(x_1))$</td>
</tr>
<tr>
<td>$x_6$</td>
<td>${{T, K_{A,TGS}, t_1}}<em>{K</em>{A,KAS}}^{*}$</td>
<td>$\pi_1(\pi_1(\pi_1(x_0)))$</td>
</tr>
<tr>
<td>$x_7$</td>
<td>${{T, K'<em>{A,TGS}, t_2}}</em>{K_{A,KAS}}^{*}$</td>
<td>$\pi_1(\pi_1(\pi_1(x_1)))$</td>
</tr>
</tbody>
</table>
Table 3. Saturation of the Frame $\phi_g$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_1 \widehat{\sigma}_g$</th>
<th>$x_1(\zeta_g)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$x_0$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$K_{\text{A,TGS}}$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>${x_0}_{x_2}$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>${x_1}_{x_2}$</td>
</tr>
<tr>
<td>$x_5$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_0)$</td>
</tr>
<tr>
<td>$x_6$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_1)$</td>
</tr>
<tr>
<td>$x_7$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_0}_{x_2})$</td>
</tr>
<tr>
<td>$x_8$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1}_{x_2})$</td>
</tr>
<tr>
<td>$x_9$</td>
<td>$t_1$</td>
<td>$\pi_2({x_0}_{x_2})$</td>
</tr>
<tr>
<td>$x_{10}$</td>
<td>$t_2$</td>
<td>$\pi_2({x_1}_{x_2})$</td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_0)})$</td>
</tr>
<tr>
<td>$x_{12}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_1)})$</td>
</tr>
<tr>
<td>$x_{13}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{14}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{15}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{16}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{17}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{18}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{19}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1({x_1(x_2)})$</td>
</tr>
<tr>
<td>$x_{20}$</td>
<td>$K_{\text{A,TGS}}$</td>
<td>$\pi_2({x_0}_{x_2})$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$K_{\text{A,TGS}}$</td>
<td>$\pi_2({x_1}_{x_2})$</td>
</tr>
<tr>
<td>$x_{22}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_2(x_3))$</td>
</tr>
<tr>
<td>$x_{23}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_2(x_3))$</td>
</tr>
<tr>
<td>$x_{24}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_2(x_3))$</td>
</tr>
<tr>
<td>$x_{25}$</td>
<td>${(A, \text{TGS}, K'<em>A, \text{TGS}, t_2)}</em>{K_{\text{A,TGS}}}^{*}$</td>
<td>$\pi_1(x_2(x_3))$</td>
</tr>
</tbody>
</table>