Deciding Knowledge Problems Modulo Classes of Permutative Theories*

Serdar Erbatur[1], Andrew M. Marshall[2], Paliath Narendran[3], and Christophe Ringeissen[4]

[1] University of Texas at Dallas, Richardson, TX, USA
serdar.erbatur@utdallas.edu
[2] University of Mary Washington, Fredericksburg, VA, USA
marshall@umw.edu
[3] University at Albany, SUNY, Albany, NY, USA
pnarendran@albany.edu
[4] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
christophe.ringeissen@loria.fr

Abstract. In the logic based approach to security protocol verification, algorithms for verifying an intruder's knowledge are critical. In this context, the capabilities of an intruder are specified by an equational theory, possibly expressed by a term rewrite system. Previous results have developed algorithms for a number of knowledge problems in many different equational and rewrite theories, such as subterm-convergent. Permutative theories such Associative-Commutative (AC) are of great interest with several procedures having been developed for AC. This leads to the question of decidability of the knowledge problems of deduction and static equivalence in permutative theories in general. It was recently shown that deduction is decidable in permutative theories. However, the decidability of static equivalence (and the related frame distinguishability problem) was still open. In this paper we show that static equivalence is undecidable in permutative theories. In addition, we show that static equivalence remains undecidable in the more restrictive case of leaf permutative theories. On the positive side, static equivalence becomes decidable for a further restricted form of permutative theories we define here.

Keywords: Permutative Equational Theories · Static Equivalence · Deduction

1 Introduction

Logic-based analysis and verification of security protocols has been a fruitful area of research, particularly for the development of formal verification tools

* Authors' version of a paper (DOI) published in *Logic-Based Program Synthesis and Transformation - 34th International Symposium, LOPSTR 2024, Milan, Italy, Sept. 9-10, 2024, Proceedings. Lecture Notes in Computer Science 14919, Springer.*

and procedures for checking various security properties of protocols, see for example [1,8,11,12,15]. In this context, the capabilities of an intruder are specified by an equational theory, possibly expressed by a term rewrite system, and the procedure seeks to verify the knowledge a potential attacker could obtain on the protocol. Two important models of intruder knowledge are, *deduction* and *static equivalence* (or frame distinguishability) [1]. One of the most common classes of equational theories is *subterm convergent term rewrite systems*, i.e., term rewrite systems where the right-hand side of the rules are ground or strict subterms of the left-hand side. For example, see the procedures developed in [1,12]. Non-orientable equalities are also useful and further results have considered specific equational theories that often arise in protocol verification. In particular, the Associative-Commutative (AC) and Commutative (C) equational theories are of great interest and have been investigated on their own, with algorithms begin developed for both the deduction and static equivalence problems, see for example [1,5,17]. Notice that the $A = \{f(x, f(y, z)) = f(f(x, y), z)\}$ (Associativity), $C = \{f(x, y) = f(y, x)\}$ (Commutativity) and AC (Associativity-Commutativity) theories are examples of permutative theories. These are theories for which the left and right side of the equality have the same number of symbols and variables (see the next section for a complete definition). Thus, the decidability results developed for permutative cases, such as AC, lead to the question of whether the knowledge problems of deduction, frame distinguishability, and static equivalence are decidable in all permutative theories. It has already been shown that deduction is decidable in permutative theories [17]. In this paper we show that for permutative theories frame distinguishability and static equivalence are undecidable. It would then be natural to consider a more restricted form of permutative theories such as leaf permutative. Notice that this is still sufficient for the AC theory because we can reformulate the associativity axiom as $f(f(x, y), z) = f(f(z, y), x)$ in order to obtain, along with commutativity, a completely leaf permuting presentation. However, we show that even when restricted to leaf permuting theories, frame distinguishability and static equivalence are still undecidable.

Another interesting reason to investigate the decidability of these knowledge problems beyond their criticality in the formal verification of protocols, is their close connection to the problems of unification and matching. Unification and matching are well established problems that arise in many applications such as logic programming and automated theorem proving. The decidability of these problems is well studied for many types of equational theories and rewrite systems. For example, it was shown in [25] that unification is undecidable in permutative theories. Later it was shown in [22] that unification is also undecidable in variable permutative theories, a subclass of permutative theories similar to the leaf permutative theories. Thus, there is a general question about the co-decidability of unification and static equivalence, and likewise matching and deduction. While we don't answer that question here we get closer by showing, like unification, static equivalence is undecidable in permutative and leaf permutative theories.

Finally, on the positive decidability side, we are able to identify a general subclass of the permutative theories for which all three knowledge problems become decidable.

Paper Outline The paper proceeds as follows: Section 2 introduces some background material and the knowledge problems under consideration in this paper. Section 3 recalls a decidability result for the deduction problem in permutative theories, and discusses the undecidability of static equivalence in permutative theories. Section 4 presents a new undecidability proof for static equivalence in leaf-permutative theories. This proof also subsumes the proof for permutative theories. Section 5 introduces a further restriction to the class of permutative theories in order to obtain decidability. Finally, Section 6 concludes the paper with a discussion, future research, and open problems.

2 Preliminaries

We use the standard notation of equational unification [7] and term rewriting systems [6]. Given a first-order signature Σ and a (countable) set of variables V, the Σ-terms over variables V are built in the usual way by taking into account the arity of each function symbol in Σ. Each Σ-term is well-formed: if it is rooted by a n-ary function symbol in Σ, then it has necessarily n direct subterms. For any term t, $|t|$ denotes the number of symbols occurring in t. The set of Σ-terms over variables V is denoted by $T(\Sigma, V)$. The set of variables from V occurring in a term $t \in T(\Sigma, V)$ is denoted by $Var(t)$. A term t is *ground* if $Var(t) = \emptyset$. For any position p in a term t (including the root position ε), $t(p)$ is the symbol at position p, $t|_p$ is the subterm of t at position p, and $t[u]_p$ is the term t in which $t|_p$ is replaced by u. A substitution is an endomorphism of $T(\Sigma, V)$ with only finitely many variables not mapped to themselves. A substitution is denoted by $\sigma = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$, where the domain of σ is $Dom(\sigma) = \{x_1, \ldots, x_m\}$ and the range of σ is $Ran(\sigma) = \{t_1, \ldots, t_m\}$. Application of a substitution σ to t is written $t\sigma$. A Σ-equation is a pair of Σ-terms denoted by $s =^? t$ or simply $s = t$ when it is clear from the context that we do not refer to an axiom.

2.1 Equational Theories

Given a set E of Σ-axioms (i.e., pairs of terms in $T(\Sigma, V)$, denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity (by a slight abuse of terminology, E is often called an equational theory). Equivalently, $=_E$ can be defined as the reflexive transitive closure $\leftrightarrow_E^*$ of an equational step $\leftrightarrow_E$ defined as follows: $s \leftrightarrow_E t$ if there exist a position p of s, $l = r$ (or $r = l$) in E, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$.

We also need to define the (sub)classes of permutative theories we consider in this paper. This is important not only for properly defining the results proven here but also because there are some previous definitions of leaf permutative theories which don't match the definition given here. For example, the definition

of leaf permutative given in this paper differs from the one given in [9] (See Definition 3 below). In [9] (Definition 3) the permutation is restricted to just the variables and is only applicable to linear terms. Thus, the definition in [9] would perhaps be better named as linear variable-permuting, while the one given here is just a permutation of the leaf-nodes of the term, see Definition 2.

Definition 1 (Permutative Theory). *An equational theory E is* permutative *if for each axiom $l = r$ in E, l and r contain the same symbols with the same number of occurrences.*

One can easily check that $A = \{f(x, f(y, z)) = f(f(x, y), z)\}$ (Associativity), $C = \{f(x, y) = f(y, x)\}$ (Commutativity) and $AC = \{f(x, f(y, z)) = f(f(x, y), z),\ f(x, y) = f(y, x)\}$ (Associativity-Commutativity) are permutative.

Two important subclasses of permutative theories are given by considering the cases where the permutations only occur on leafs or on variables.

Definition 2 (Leaf Permutative Theory). *An equational theory E is* Leaf permutative *if for each axiom $l = r$ in E, r is a leaf permutation of l, i.e., $r = l\sigma$, where σ is a permutation of the leaf nodes of l.*

For example, C is leaf permutative, but A is not.

Definition 3 (Variable-Permuting Theory). *An axiom $l = r$ is said to be* variable-permuting *[22] if all the following conditions are satisfied:*

1. *the set of occurrences of l is identical to the set of occurrences of r,*
2. *for any non-variable occurrence p of l, $l(p) = r(p)$,*
3. *for any $x \in Var(l) \cup Var(r)$, the number of occurrences of x in l is identical to the number of occurrences of x in r.*

Definition 4 (Shallow Theory). *An axiom $l = r$ is* shallow *if variables can only occur at a position at depth at most 1 in both l and r. An equational theory is* shallow *if all its axioms are shallow.*

For example, C is shallow and permutative, but A is not. Note that a shallow theory is not necessarily permutative. For example, $\{x + 0 = x\}$ is shallow but not permutative.

2.2 Rewrite Relations

Given a signature Σ, an oriented Σ-axiom is called a rewrite rule of the form $l \to r$ such that $l, r \in T(\Sigma, V)$, l is not a variable and $Var(r) \subseteq Var(l)$. A finite set of rewrite rules is called a *term rewriting system* (TRS, for short). Let R be any TRS. For any Σ-terms s and t, s *R-rewrites* to t, denoted by $s \to_R t$, if there exist a position p of s, $l \to r \in R$, and substitution σ such that $s_{|p} = l\sigma$ and $t = s[r\sigma]_p$. The term s is said to be *R-reducible*, $s_{|p}$ is called a *redex*, and in the particular case where $s_{|p} = l\sigma$, s *R-rewrites* to t, denoted by $s \to_R t$. A TRS R is *terminating* if there are no infinite rewriting sequences with respect

to $\rightarrow_R$. A term is an *innermost redex* if none of its proper subterms is a redex. The symmetric relation $\leftarrow_R \cup \rightarrow_R$ is denoted by $\longleftrightarrow_R$. The rewrite relation $\rightarrow_R$ is confluent if $\longleftrightarrow_R^*$ is included in $\rightarrow_R^* \circ \leftarrow_R^*$. The rewrite relation $\rightarrow_R$ is *convergent* if $\rightarrow_R$ is both terminating and convergent. When $\rightarrow_R$ is convergent, we have that for any terms t, t', $t \longleftrightarrow_R^* t'$ iff $t{\downarrow_R} = t'{\downarrow_R}$, where $t{\downarrow_R}$ (resp., $t'{\downarrow_R}$) denotes the unique normal form of t (resp., t') w.r.t $\rightarrow_R$.

Definition 5 (Unification). *Given a signature Σ, an equational theory E, and a set of Σ-equations, $\Gamma = \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$. The E-unification decision problem asks if there exists a substitution σ such that $s_i\sigma =_E t_i\sigma$ for all $1 \leq i \leq n$. If E is presented as a rewrite relation R, then we ask if there exists a substitution σ such that $s_i\sigma \leftrightarrow_R^* t_i\sigma$ for all $1 \leq i \leq n$.*

2.3 Knowledge Problems

The applied pi calculus and frames are used to model attacker knowledge [2]. In this model, the set of messages or terms which the attacker knows, and which could have been obtained from observing one or more protocol sessions, are the set of terms in $Ran(\sigma)$ of the frame $\phi = \nu\tilde{n}.\sigma$, where σ is a substitution ranging over ground terms. We also need to model cryptographic concepts such as nonces, keys, and publicly known values. We do this by using names, which are essentially free constants. Here also, we need to track the names which the attacker knows, such as public values, and the names which the attacker does not know a priori, such as freshly generated nonces. $\tilde{n}$ consists of a finite set of restricted names; these names represent freshly generated names which remain secret from the attacker. The set of names occurring in a term t is denoted by $fn(t)$. For any frame $\phi = \nu\tilde{n}.\sigma$, let $fn(\phi)$ be the set of names $fn(\sigma)\backslash\tilde{n}$ where $fn(\sigma) = \bigcup_{t \in Ran(\sigma)} fn(t)$; and for any term t, let $t\phi$ denote — by a slight abuse of notation — the term $t\sigma$. For any term t, we say that t satisfies the name restriction of ϕ if $fn(t) \cap \tilde{n} = \emptyset$.

Definition 6 (Deduction). *Let $\phi = \nu\tilde{n}.\sigma$ be a frame, and t a ground term. We say that t is* deduced from ϕ modulo E*, denoted by $\phi \vdash_E t$, if there exists a term ζ such that $\zeta\sigma =_E t$ and $fn(\zeta) \cap \tilde{n} = \emptyset$. The term ζ is called a* recipe *of t in ϕ modulo E.*

Notice that deduction is modeling the ability of an adversary to deduce some secret, that is supposed to remain hidden, from a protocol. This is a critical measure of security for key establishment protocols. Another measure of security attempts to see if an adversary could tell two different runs of the same protocol apart. This security requirement is not modeled by deduction but is important for voting protocols. In such protocols you don't want an observer to be able to tell a vote for one candidate from another based on watching the protocol for multiple runs. This form of security is modeled by *statically equivalent* modulo E.

Definition 7 (Static Equivalence). *Two terms s and t are* equal *in a frame $\phi = \nu\tilde{n}.\sigma$ modulo an equational theory E, denoted $(s =_E t)\phi$, if $s\sigma =_E t\sigma$, and $\tilde{n} \cap (fn(s) \cup fn(t)) = \emptyset$. The set of all equalities $s = t$ such that $(s =_E t)\phi$ is denoted by $Eq(\phi)$. Given a set of equalities Eq, the fact that $(s =_E t)\phi$ for all $s = t \in Eq$ is denoted by $\phi \models Eq$. Two frames $\phi = \nu\tilde{n}.\sigma$ and $\psi = \nu\tilde{n}.\tau$ are* statically equivalent modulo E, *denoted as $\phi \approx_E \psi$, if $Dom(\sigma) = Dom(\tau)$, $\phi \models Eq(\psi)$ and $\psi \models Eq(\phi)$.*

There is a closely related problem, that asks if there exists a pair of recipe terms that distinguish two different frames.

Definition 8 (Frame Distinguishability). *Given frames $\phi = \nu\tilde{n}.\sigma$ and $\psi = \nu\tilde{n}.\tau$, we say that ϕ is distinguishable from ψ in theory E, denoted $\phi \not\approx_E \psi$, if there exist two terms, t and s (with $\tilde{n} \cap (fn(s) \cup fn(t)) = \emptyset$), such that $t\sigma =_E s\sigma$ and $t\tau \neq_E s\tau$.*

Notice that in this paper we are considering the decision problem form of each of the knowledge problems listed above. Thus, if an algorithm for frame distinguishability returns true for an pair of frames, we know that the frames are not static equivalent. Likewise, if and algorithm for static equivalence returns true for a pair of frames, we know that there is no pair of recipes that distinguish the frames, i.e., frame distinguishability is false. Now, if were to considered the form of the frame distinguishability problem that requires the algorithm to produce the actual witness pair of recipes then the proof given in Theorem 3 would still show the undecidability of this form of the problem. However, the relation to the static equivalence problem would be a little less clear since, since if a static equivalence algorithm returns false, we know at least of pair of witness recipes exists but to find them would possible require searching the sets $Eq(\phi)$ and $Eq(\psi)$.

Finally we can note that the above knowledge problems, deduction, static equivalence, and frame distinguishable, are known to be decidable for subterm convergent rewrite systems [1].

2.4 Turing Machines and Linear Bounded Automata

We need to recall some standard background material on Turing Machines (TMs) and Linear Bounded Automata (LBA) for the later undecidability results; see [19] for a more complete background. We will represent a TM, M, as a 7-tuple, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the tape alphabet. $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$, where $\sqcup$ represents the blank symbol. $\delta : (Q \setminus \{q_a, q_r\} \times \Gamma) \longrightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function, where R represents a right move and L a left move. $q_0 \in Q$ is the unique initial state, $q_a \in Q$ is the unique accept state, and $q_r \in Q$ is the unique reject state. LBA are Turing Machines with a tape that is bounded by the size of the input string (plus two tape end-caps) [20,18]. That is, we introduce two new tape symbols, $\{<, >\} \in \Gamma$. We restrict δ so that it cannot move left of $<$ or right of $>$, $\delta : (Q \setminus \{q_a, q_r\} \times <) = (Q \times < \times \{R\})$ and

$\delta : (Q \setminus \{q_a, q_r\} \times >) = (Q \times > \times \{L\})$. The initial configuration of a LBA on input word $w \in \Sigma^*$ is in state q_0, the read-write head over the left end-cap $<$ and the tape containing $< \cdot w \cdot >$. See Example 1 in Section 4 for a simple example.

3 Decidability of Knowledge Problems in Permutative Theories

In this section we consider the decidability of both knowledge problems in classes of permutative theories. In particular we consider permutative theories and leaf permutative theories.

3.1 Decidability of Deduction in Permutative Theories

It has already been shown in [10] that deduction is decidable in permutative theories.

Theorem 1 ([10]). *Deduction is decidable for any permutative theory.*

This is due to the fact that you can put a bound on the number of terms you need consider since permutative theories are non-size reducing. However, when considering static equivalence, the problem becomes more difficult. We briefly consider permutative theories in the next section.

3.2 Undecidability of Frame Distinguishability in Permutative Theories

While deduction is decidable it happens that frame distinguishability, and thus static equivalence, are undecidable in permutative theories.

Theorem 2. *Frame Distinguishability is undecidable in general for permutative theories.*

However, this result is subsumed by the result in the next section that shows that frame distinguishability is also undecidable in the more restrictive leaf permutative theories. Thus, we present the proof details for that more restrictive case in the following section. However, one could prove the permutative case directly by starting with the method developed in [25] for proving that the unification problem is undecidable in general for permutative theories. The key to the proof is to show how for an arbitrary deterministic Turing Machine (TM for short), M, a permutative and confluent TRS, R_M, could be created that simulates M. By starting with a suitable undecidable TM problem, one could use the construction from [25] to reduce to the frame distinguishability problem.

4 Static Equivalence in Leaf Permuting Theories

In this section we prove that the frame distinguishability problem, and thus static equivalence, is undecidable for leaf permuting theories. The results proceeds as follows, we first develop an undecidable result for Linear Bounded Automata (LBA) which we will use in the reductions. Next, we show how to create a leaf permuting TRS from a LBA. Finally, we use this TRS to show that the frame distinguishability problem is undecidable.

Lemma 1. *Given an arbitrary LBA, M, with input alphabet Σ, it's undecidable if there exists a string, $w \in \Sigma^*$, such that M accepts w.*

Proof. Easy reduction from the LBA empty language problem proved undecidable in [3]. There it is shown that it is undecidable if $L(M) = \emptyset$ for an arbitrary deterministic LBA M.

Next, we need to show how to obtain a leaf permuting TRS from a LBA.

Lemma 2. *Given a deterministic LBA, M, one can construct a leaf permuting TRS R such that if M accepts a string w then there exists a term t which encodes the initial configuration of M on input w and a term s that encodes the final accepting configuration of M on w such that $t \downarrow_R = s$.*

Proof. Here we modify the encoding from [25] to obtain a conversion from LBA to leaf permuting TRS. Let $M = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$. Assume $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and $\Gamma = \{<, >, \sqcup\} \cup \Sigma$, where $<, >$ are the left and right end caps respectively, and $\sqcup$ is the blank symbol. We now need to construct the terms that will represent the tape of the LBA. We introduce three new non-constant function symbols, f, g, h and three new constants, a, b, and P. We use each as follows:

- h has arity $|Q|+1$ and is used to represent the state of the LBA. To represent state q_i, a constant b is placed at the i^{th} position with the remaining positions containing a constants. For example, if $|Q| = 2$ then q_0 is represented as $h(b, a, a)$. The final configuration, with constant b in the final position, is used to represent a non-state or "dummy state", the use of which is described below. We denote this dummy state as q_d.
  - **We use the notation $h(q_i)$ to abbreviate the encoding of the state q_i using h.**
- g has arity $|\Gamma|$ and is used to encode the alphabet characters. We place a constant b at position i in g with a constants placed at all other positions to encode a_i. Positions $n+1, \ldots n+3$ are used to encode $\{<, >, \sqcup\}$ in the same way.
  - **We use the notation $g(a_i)$ to abbreviate the encoding of the character a_i using g.**

– f is used to form terms which consist of an encoding of a state, an encoding of a single character, and an f-rooted subterm or the constant P. The subterm is used to encode the rest of the string. The constant P is used to stop the encoding. For example, the dummy state and the character a_i can be encoded as the term $f(h(q_d), g(a_i), P)$.

We now form terms representing the configurations of the LBA and its tape as follows:

– For any string $w \in \Gamma^+$ we represent w as a layered f-term, one layer per alphabet character. The last layer is ended using the P constant. The dummy state is used by default for each of the state positions in the f-terms.
  - **We use the notation $f(\overline{w})$ to abbreviate the encoding of the string w using f-terms**.
  - For example, let $<w_1 q_i a_i w_2>$ be a configuration of the LBA. We represent this by an f rooted term as $f(\overline{<w_1}, f(h(q_i), g(a_i), f(\overline{w_2>})))$.

We now need to construct the leaf permuting TRS. Let's consider the moves of the transition function, δ, and construct from them a TRS R:

– For each right move, $\delta(q_i, a_i) = (q_j, a_j, R)$, we create a rule for each possible character $a_k \in \Gamma \setminus \{>\}$ of the form:

$$f(h(q_i), g(a_i), f(h(q_d), g(a_k), x)) \to f(h(q_d), g(a_j), f(h(q_j), g(a_k), x))$$

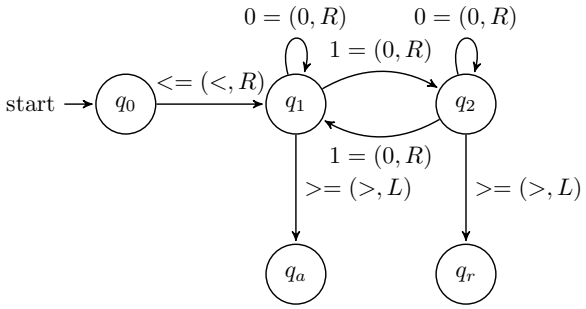
– For each left move, $\delta(q_i, a_i) = (q_j, a_j, L)$, we create a rule for each possible character $a_k \in \Gamma \setminus \{<\}$ of the form:

$$f(h(q_d), g(a_k), f(h(q_i), g(a_i), x)) \to f(h(q_j), g(a_k), f(h(q_d), g(a_j), x))$$

Finally, we need to describe the initial configurations for the LBA. The LBA will start in the configuration $q_o < w >$ for input w. We encode this as an f-term, where all the states are initially $h(q_d)$ accept the first (most left h). That is, we encode using the term $t = f(h(q_0), g(<), f(\overline{w>}))$.

Notice that every rule in R is leaf permuting. In addition, the LBA accepts the string w iff $f(h(q_0), g(<), f(\overline{w>})) \to_R^* s$ such that s is a term that encodes a final accepting configuration of the LBA. This final configuration will have just a single state $h(q_a)$ with all other state positions being dummy states $h(q_i)$.

Example 1. Let's consider the following toy LBA and construct the corresponding TRS.



Notice for this simple example $\Gamma = \{<, >, 0, 1, \sqcup\}$ and $Q = \{q_0, q_1, q_2, q_r, q_a\}$. To encode the symbol $<$ we use $g(b, a, a, a, a)$, and 0 is $g(a, a, b, a, a)$. Now to encode the state q_0 we use $h(b, a, a, a, a, a)$, and we use $h(a, b, a, a, a, a)$ for q_1. Notice the additional final position in $h()$, this is to encode the dummy, q_d, state that is just used to track the current state of the LBA during rewriting, and we encode this state with $h(a, a, a, a, a, b)$.

Now let's construct one of the rewrite rules based on the above transition function in full without any of the shorthand developed above. We can then construct several others using the shorthand. Consider $\delta(q_0, <) = (q_1, <, R)$ in TRS rule form one of the instances of this rule would be:

$$f(h(b,a,a,a,a,a), g(b,a,a,a,a), f(h(a,a,a,a,a,b), g(a,a,b,a,a), x)) \rightarrow \\ f(h(a,a,a,a,a,b), g(b,a,a,a,a), f(h(a,b,a,a,a,a), g(a,a,b,a,a), x))$$

Notice that this is for the rule where 0 (encoded as $g(a, a, b, a, a)$) is on the right of $<$ on the tape. We would also need the same rule for each of the other symbols.

Let's construct some of the rewrite rules for the above LBA using the more compact notation. for the transition $\delta(q_0, <) = (q_1, <, R)$ some of the rules are:

$$f(h(q_0), g(<), f(h(q_d), g(0), x)) \rightarrow f(h(q_d), g(<), f(h(q_1), g(0), x)) \\ f(h(q_0), g(<), f(h(q_d), g(>), x)) \rightarrow f(h(q_d), g(<), f(h(q_1), g(>), x)) \\ f(h(q_0), g(<), f(h(q_d), g(1), x)) \rightarrow f(h(q_d), g(<), f(h(q_1), g(1), x))$$

For the transition $\delta(q_2, 1) = (q_1, 0, R)$, some of the resulting rules are:

$$f(h(q_2), g(1), f(h(q_d), g(1), x)) \rightarrow f(h(q_d), g(0), f(h(q_1), g(1), x)) \\ f(h(q_2), g(1), f(h(q_d), g(0), x)) \rightarrow f(h(q_d), g(0), f(h(q_1), g(0), x))$$

For the transition $\delta(q_2, >) = (q_r, >, L)$, some of the resulting rules are:

$$f(h(q_d), g(0), f(h(q_2), g(>), x)) \rightarrow f(h(q_r), g(0), f(h(q_d), g(>), x)) \\ f(h(q_d), g(1), f(h(q_2), g(>), x)) \rightarrow f(h(q_r), g(1), f(h(q_d), g(>), x))$$

Finally let's show an encoding of a string and a rewrite step. Assume we have an initial configuration of $q_0 <01>$. This configuration is encoded as

$$f(h(q_0), g(<), f(h(q_d), g(0), f(h(q_d), g(1), f(h(q_d), g(>), P))))$$

Notice that for any configuration encoding there is always only one position with a non-dummy state. For this term we can apply the rule

$$f(h(q_0), g(<), f(h(q_d), g(0), x)) \rightarrow f(h(q_d), g(<), f(h(q_1), g(0), x))$$

which results in the rewrite step:

$$f(h(q_0), g(<), f(h(q_d), g(0), f(h(q_d), g(1), f(h(q_d), g(>), P)))) \rightarrow_R \\ f(h(q_d), g(<), f(h(q_1), g(0), f(h(q_d), g(1), f(h(q_d), g(>), P))))$$

Notice that since the LBA is deterministic we obtain the following result.

Lemma 3. *Let M be a deterministic LBA. Then the variable permuting TRS, R, constructed from M is locally confluent. Furthermore, if M is both deterministic and terminating then the rewrite sequence $t \rightarrow_R^* s$, terminates when t is the term encoding of the start configuration for M on some input string w.*

We now need to describe the LBA and TRS we will use in the Reduction. First, from a LBA M_1 we can easily construct the following LBA.

Lemma 4. *Let M_1 be a LBA that before accepting and halting it replaces the tape (except the end caps) with blank symbols and stops (enters the accept or reject state) with the tape head over the left end-cap. One can construct a LBA M_2 from M_1 such that $L(M_2) = \emptyset$ and every transition of M_1 and M_2 are the same except the accepting transitions from M_1 are now rejecting in M_2.*

Notice that for each LBA M_1 and M_2 there is a corresponding leaf permuting TRS, R_1 and R_2 respectively. However, we need to combine these two TRS into a single TRS.

Lemma 5. *Given LBAs M_1 and M_2 and their leaf permutative TRSs R_1 and R_2 respectively. Let q_{0_i} be the initial state for M_i. One can construct a leaf permutative TRS $R_{1,2}$ such that for input string w and term $t_i = f(h(q_{0_i}), g(<), f(\overline{w>}))$, $t_i \rightarrow_{R_{1,2}}^* s_i$ iff $t_i \rightarrow_{R_i}^* s_i$, $i \in \{1, 2\}$.*

Proof. One just needs to ensure that $\{Q_1 \setminus \{q_a, q_r\}\} \cap \{Q_2 \setminus \{q_a, q_r\}\} = \emptyset$. Then, the rules of R_1 and R_2 are disjoint.

We can now combine all the above into the final undecidability proof.

Theorem 3. *Frame distinguishability is undecidable for leaf permuting theories.*

Proof. We can proceed by reduction using Lemma 1. Assume we are given a deterministic LBA M_1. Without loss of generality assume that before halting and entering q_a or q_r, M_1 erases its tape and halts with the head over the left end-cap. Furthermore, assume that as an initial step M_1 scans the tape from left to right end-cap and then back to the left. Finally, we can assume q_0 is the unique start state of M_1 and after leaving this state the LBA never returns to it. Since these steps could be added to any LBA without changing its language, they don't represent a restriction. We proceed as follows.

1. From M_1 construct the always reject LBA M_2 as in Lemma 4. We can assume w.l.o.g., that $\{Q_1 \setminus \{q_a, q_r\}\} \cap \{Q_2 \setminus \{q_a, q_r\}\} = \emptyset$. This can be done by simply creating a marked version of $Q_1 \setminus \{q_a, q_r\}$, s.t. a $q_{i_2} \in Q_2$ for each $q_{i_1} \in Q_1$. Let q_{0_i} be the start state for M_i.
   - For any transition from M_1, $\delta(q_{i_1}, a_i) = (q_{j_1}, a_j, x)$, if $q_{j_1} \neq q_a$ then for M_2, $\delta(q_{i_2}, a_i) = (q_{j_2}, a_j, x)$. If $q_{j_1} = q_a$ then $\delta(q_{i_2}, a_i) = (q_r, a_j, x)$.
2. From M_1 and M_2 construct R_1 and R_2 as in Lemma 2.

- Notice if a term t_1 encodes a configuration and $t_1 \rightarrow^+_{R_1} t_2$ s.t. t_2 doesn't encode an accepting configuration, then $t'_1 \rightarrow^+_{R_2} t'_2$, where $t_i = t'_i$ except each state q_{i_1} is swapped for q_{i_2}.

3. Construct $R_{1,2}$ from R_1 and R_2 as in Lemma 5.
4. Construct two frames, such that $\tilde{n} = \emptyset$ for each frame::

a) $\phi_1 = \nu\tilde{n}.\sigma_1 = \nu\tilde{n}.\{x \mapsto h(q_{0_1})\}$ b) $\phi_2 = \nu\tilde{n}.\sigma_2 = \nu\tilde{n}.\{x \mapsto h(q_{0_2})\}$

Assume that we have an algorithm for the frame distinguishability problem and it returns true. Then, there exists a recipe pair (t, s) such that $t\sigma_1 \leftrightarrow^*_{R_{1,2}} s\sigma_1$ and $t\sigma_2 \not\leftrightarrow^*_{R_{1,2}} s\sigma_2$. We can assume the following about these two recipe terms:

(i) It can't be that $s = t$, nor can t and s both be ground, otherwise $t\sigma_2 \leftrightarrow^*_{R_{1,2}} s\sigma_2$. Thus, $t\sigma_1 \leftrightarrow^+_{R_{1,2}} s\sigma_1$.

(ii) For at least one of the terms, t and s, we have that $t\{x \mapsto h(q_{0_1})\} \neq t$ and (or) $s\{x \mapsto h(q_{0_1})\} \neq s$. That is, we use the substitution on x. If this was not the case, then $t\sigma_2 \leftrightarrow^*_{R_{1,2}} s\sigma_2$.

(iii) At least one of, $s\sigma_1$ (and $s\sigma_2$) or $t\sigma_1$ (and $t\sigma_2$), encodes a valid, single, starting configuration, and this starting configuration is the entire term (not a subterm of a larger term). This is due to the following (we consider $t\sigma_i$ but the same applies for $s\sigma_i$):
   - Notice the rules of $R_{1,2}$ can only be applied to redexes encoding valid configurations and we have that $t\sigma_1 \leftrightarrow^+_{R_{1,2}} s\sigma_1$. Since M_1 (and thus M_2) first scan the tape from left to right, the rewrite derivation on $t\sigma_1$ would stop when any malformed subterm was reached and before the final configuration. However, since the steps of M_2, except the step entering the final configuration, are the same as those for M_1, M_2 would stop at the same point in the derivation from $t\sigma_2$. That is, $t\sigma_2 \rightarrow^*_{R_{1,2}} s\sigma_2$. Thus, $t\sigma_i$ must contain a valid configuration.
   - Suppose all the redex for the rewrite derivation $t\sigma_i \leftrightarrow^*_{R_{1,2}} s\sigma_i$ are contained in a proper subterm of t, t'. Then we could just consider the recipe pair (t', s). Likewise, if t encodes multiple disjoint configurations as subterms, then there must be at least one of these subterms, say t', such that $t'\sigma_1 \leftrightarrow^*_{R_{1,2}} s\sigma_1$ and $t'\sigma_2 \not\leftrightarrow^*_{R_{1,2}} s\sigma_2$, otherwise $t\sigma_2 \leftrightarrow^*_{R_{1,2}} s\sigma_2$. Thus, we could just consider the recipe pair (t', s). Thus, $t\sigma_i$ encodes a single valid configuration which is the entire term.
   - $t\sigma_i$ encodes an initial configuration since, by construction, q_{0_i} is only used for the initial starting configuration of M_i and never reused. Thus, if $t\sigma_i \neq t$ then $t\sigma_i$ encodes an initial configuration. Therefore, $t\sigma_i$ encodes a single valid initial configuration.

Let $t\sigma_1 \rightarrow^*_{R_{1,2}} r \leftarrow^*_{R_{1,2}} s\sigma_1$. Without loss of generality we can assume the following for this derivation:

(a) r encodes an accepting configuration, if not then $t\sigma_2 \rightarrow^*_{R_{1,2}} r \leftarrow^*_{R_{1,2}} s\sigma_2$.

Notice that (a) allows us to simplify the rewrite derivation. Without loss of generality we can assume that $s\sigma_i = r$. Since r must be an accepting configuration and at least one side of the derivation $t\sigma_i \rightarrow^*_{R_{1,2}} r \leftarrow^*_{R_{1,2}} s\sigma_i$ must use the substitution on x, we can assume that side to be $t\sigma_i \rightarrow^*_{R_{1,2}} r$ (swap if not). Now the pair (t, r) form a recipe pair s.t. $t\sigma_1 \leftrightarrow^*_{R_{1,2}} r\sigma_1$ and $t\sigma_2 \not\leftrightarrow^*_{R_{1,2}} r\sigma_2$. Therefore, we can assume that $t\sigma_1 \rightarrow^+_{R_{1,2}} s\sigma_1$ and $t\sigma_2 \not\rightarrow^+_{R_{1,2}} s\sigma_2$. Finally, from (i) - (iii) above, $t\sigma_i$ is a well formed term encoding an initial configuration of an LBA M_i for some initial input string w. From Lemma 2, $t\sigma_1 \rightarrow^+_{R_{1,2}} s\sigma_1$ iff M_1 accepts some string w.

Therefore, a frame distinguishability algorithm would allow us to decide if for an arbitrary LBA M_1 if there exists some string w, accepted by M_1, violating Lemma 1.

If the frame distinguishability problem is undecidable then we also get undecidability of the static equivalence problem and since distinguishability is undecidable for leaf permutative theories it is also undecidable for the more general permutative theories.

Corollary 1. *Frame distinguishability is undecidable for permutative theories.*

5 Decidable Static Equivalence in Subclasses of Permutative Theories

A decision procedure for static equivalence in some permutative theories E can obtained by computing a finite set of equalities bounded by $|E|$, where $|E| = \max_{\{l \mid l=r \in E\}} |l|$. To define this particular finite set of equalities, we first construct a frame saturation dedicated to permutative theories. Then, the recipes of the (deducible) terms included in that saturation are used to build an appropriate set of bounded equalities. Given a term t, $St(t)$ is the smallest set of terms including t such that

- if $u' =_E u$ and $u \in St(t)$, then $u' \in St(t)$,
- if $u \in St(t)$ and p is a non-root position of u, then $u|_p \in St(t)$.

Notice that $St(t)$ is finite since E is permutative. For a set of terms T, $St(T) = \bigcup_{t \in T} St(t)$, and for a substitution σ, $St(\sigma) = St(Ran(\sigma))$.

Definition 9 (Frame Saturation for Permutative Theories). *Let E be a permutative theory. For any frame $\phi = \nu\tilde{n}.\sigma$, let $ESt(\sigma) = St(\sigma) \cup \bigcup_{t \in ET} St(t)$ where ET is the set of terms $s[\boldsymbol{d}]$ such that $|s| \leq |E|$, $fn(s) \cap \tilde{n} = \emptyset$, $\boldsymbol{d} \in St(\sigma)$ and $s[\boldsymbol{d}] \leftrightarrow^{\epsilon}_E t'$ for some term t'.*

The set of terms $sat_E(\phi)$ is the smallest set D such that:

(1) $Ran(\sigma) \subseteq D$,
(2) *if* $t_1, \ldots, t_n \in D$ *and* $f(t_1, \ldots, t_n) \in ESt(\sigma)$ *then* $f(t_1, \ldots, t_n) \in D$,
(3) *if* $t \in D$, $t' \in ESt(\sigma)$, $t =_E t'$, *then* $t' \in D$,

For any frame $\phi = \nu\tilde{n}.\sigma$, σ_* denotes the substitution defined as follows:

$$\sigma\{\chi_u \mapsto u \mid u \in sat_E(\phi)\backslash Ran(\sigma)\}$$

where χ_u is a fresh variable for any $u \in sat_E(\phi)\backslash Ran(\sigma)$, and ϕ_* denotes the frame $\nu\tilde{n}.\sigma_*$. Given a recipe ζ_u for each $u \in sat_E(\phi)\backslash Ran(\sigma)$, the substitution $\{\chi_u \mapsto \zeta_u \mid u \in sat_E(\phi)\backslash Ran(\sigma)\}$ is called a recipe substitution *of ϕ and is denoted by ζ_ϕ.*

For any set Eq of equalities between two terms satisfying the name restriction of ϕ, we denote $\phi \models Eq$ if for any $t = t' \in Eq$ we have $t\phi =_E t'\phi$. The set $Eq^B(\phi)$ is the set of equalities $t\zeta_\phi = t'\zeta_\phi$ such that t, t' are terms satisfying the name restriction of ϕ, $|t| \leq |E|$, $|t'| \leq |E|$ and $(t\zeta_\phi)\phi =_E (t'\zeta_\phi)\phi$.

Definition 10. *A permutative theory E is said to be* locally stable permutative *(LSP, for short), if the following property holds: for any frame ϕ and any terms s and t satisfying the name restriction of ϕ, if $s\phi_* =_E t\phi_*$ and $\psi \models Eq^B(\phi)$ then $(s\zeta_\phi)\psi =_E (t\zeta_\phi)\psi$.*

LSP theories bear similarities with locally stable theories as defined in [1]. In the context of this paper, we consider a slight adaptation of local stability by assuming a fixed finite set of deducible terms for all permutative theories we want to target. Then, Definition 10 captures the main property to satisfy in order to get local stability and then a decision procedure for static equivalence based on checking finitely many equalities.

Lemma 6 ([17]). *Any shallow permutative theory is LSP.*

Remark 1. Unification in shallow permutative theories is decidable and finitary since there exists a sound, complete and terminating unification procedure for shallow theories [14].

Particular variable-permuting theories provide other examples of *LSP* theories.

Definition 11 (Separate Variable-Permuting Theory). *An equational theory E is said to be* separate variable-permuting *(SVP, for short) if E is variable-permuting and for any $l = r \in E$, l and r are rooted by the same function symbol that does not occur elsewhere in E.*

Lemma 7 ([10]). *Any SVP theory is LSP.*

Remark 2. Unification in *SVP* theories is decidable and finitary since all *SVP* theories are closed by paramodulation and there exists a sound, complete and terminating unification procedure for theories closed by paramodulation [23,21].

Example 2. Consider

$$K = \{keyexch(x, pk(x'), y, pk(y')) = keyexch(x', pk(x), y', pk(y))\}.$$

K is a permutative theory which is not shallow. However, it is a *SVP* theory, and so it is a *LSP* theory.

The equational theory K, combined with a subterm convergent term rewrite system, is useful in practice to model a group messaging protocol [13]. One can notice that the pk symbol occurs in K as an "inner" constructor. Thus, K can be combined with a term rewrite system R sharing pk, provided that pk is a constructor of R [16]. Consequently, variable-permuting theories like K provide good examples to constructor-sharing combination. The application of shallow permutative theories to constructor-sharing combination is quite limited: in that case, the inner constructors can only occur in ground terms. For this reason, it is clearly interesting to identity *LSP* theories beyond the case of shallow permutative. We conjecture the existence of *LSP* theories that are neither shallow permutative nor variable-permuting.

A decision procedure for static equivalence in *LSP* theories is given by the following lemma:

Lemma 8. *Let E be any LSP theory. For any frames ϕ and ψ, $\phi \approx_E \psi$ iff $\psi \models Eq^B(\phi)$ and $\phi \models Eq^B(\psi)$.*

Proof. (If direction) Let $Eq(\phi)$ be the set of all equalities $s\zeta_\phi = t\zeta_\phi$ such that $(s\zeta_\phi =_E t\zeta_\phi)\phi$.

Consider any $s\zeta_\phi = t\zeta_\phi \in Eq(\phi)$. By definition of ζ_ϕ and ϕ_*, we have $(s\zeta_\phi)\phi =_E s\phi_*$ and $(t\zeta_\phi)\phi =_E t\phi_*$. Since $(s\zeta_\phi)\phi =_E (t\zeta_\phi)\phi$, we obtain $s\phi_* =_E t\phi_*$. Then, by Definition 10, we get $(s\zeta_\phi)\psi =_E (t\zeta_\phi)\psi$, which means that $\psi \models Eq(\phi)$. In a symmetric way, we show that $\phi \models Eq(\psi)$. Then, we can conclude since $\phi \approx_E \psi$ iff $\psi \models Eq(\phi)$ and $\phi \models Eq(\psi)$.

Theorem 4. *Static equivalence is decidable in any LSP theory.*

6 Conclusion

We have shown in this paper that static equivalence is undecidable in permutative theories and that it remains undecidable even if one restricts the equational theory to just leaf permutative. It is interesting to note that the problem of unification is also undecidable in these theories and has a close connection to static equivalence by definition. Thus, it would be interesting to know the relation between these two problems with respect to decidability. There are a number of results on this question already, For example it's shown in [2] that both static equivalence and deduction are undecidable in general and that there are theories for which deduction is decidable but static equivalence is undecidable. It's shown in [10] that deduction is decidable in permutative theories but we have in [22] that unification is undecidable for both permutative and variable permutative theories. Interestingly, in [4] a theory is developed for which unification is decidable but deduction is undecidable. Perhaps it would be possible to extend this result in order to show that static equivalence is also undecidable.

With respect to positive results, we have also identified a class of permutative theories for which all knowledge problems are decidable.

We plan to continue the study of the knowledge problems in equational theories given by rewrite systems modulo permutative axioms. On the one hand, it is possible to consider extensions of subterm rewrite systems, such as contracting rewrite systems [24,10]. On the other hand, static equivalence is decidable in particular permutative theories defined by shallow permutative axioms or by some specific "disjoint" variable-permuting axioms. We are working on finding additional permutative theories with decidable static equivalence. Due to the undecidability result reported here, it is clear now that we cannot consider any arbitrary set of permutative axioms.

References

1. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. **367**(1-2), 2–32 (2006)
2. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Hankin, C., Schmidt, D. (eds.) Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001. pp. 104–115. ACM (2001)
3. Akçam, Z., Cornell, K.A., Hono II, D.S., Narendran, P., Pulver, A.: On problems dual to unification: The string-rewriting case (2023), available at doi.org/10.48550/arXiv.2103.00386
4. Anantharaman, S., Narendran, P., Rusinowitch, M.: Intruders with caps. In: Baader, F. (ed.) Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4533, pp. 20–35. Springer (2007)
5. Ayala-Rincón, M., Fernández, M., Nantes-Sobrinho, D.: Intruder deduction problem for locally stable theories with normal forms and inverses. Theor. Comput. Sci. **672**, 64–100 (2017)
6. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York, NY, USA (1998)
7. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 445–532. Elsevier and MIT Press (2001)
8. Baudet, M., Cortier, V., Delaune, S.: YAPA: A generic tool for computing intruder knowledge. ACM Trans. Comput. Log. **14**(1), 4 (2013)
9. Boy De La Tour, T., Echenim, M., Narendran, P.: Unification and matching modulo leaf-permutative equational presentations. In: International Joint Conference on Automated Reasoning. pp. 332–347. Springer (2008)
10. Bunch, C., Satterfield, S.D., Erbatur, S., Marshall, A.M., Ringeissen, C.: Knowledge problems in protocol analysis: Extending the notion of subterm convergent (2024), available at doi.org/10.48550/arXiv.2401.17226
11. Chadha, R., Cheval, V., Ciobâcă, S., Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. ACM Trans. Comput. Log. **17**(4), 23:1–23:32 (2016)
12. Ciobâcă, S., Delaune, S., Kremer, S.: Computing knowledge in security protocols under convergent equational theories. J. Autom. Reasoning **48**(2), 219–262 (2012)

13. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1802–1819. ACM (2018)
14. Comon, H., Haberstrau, M., Jouannaud, J.: Syntacticness, cycle-syntacticness, and shallow theories. Inf. Comput. **111**(1), 154–191 (1994)
15. Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond subterm-convergent equational theories in automated verification of stateful protocols. In: Maffei, M., Ryan, M. (eds.) Principles of Security and Trust. pp. 117–140. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
16. Erbatur, S., Marshall, A.M., Ringeissen, C.: Notions of knowledge in combinations of theories sharing constructors. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, Proceedings. LNCS, vol. 10395, pp. 60–76. Springer (2017)
17. Erbatur, S., Marshall, A.M., Ringeissen, C.: Computing knowledge in equational extensions of subterm convergent theories. Math. Struct. Comput. Sci. **30**(6), 683–709 (2020)
18. Hopcroft, J.E., Ullman, J.D.: Some results on tape-bounded Turing machines. Journal of the ACM **16**(1), 168–177 (1969)
19. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
20. Kuroda, S.Y.: Classes of languages and linear-bounded automata. Information and control **7**(2), 207–223 (1964)
21. Lynch, C., Morawska, B.: Basic syntactic mutation. In: Voronkov, A. (ed.) Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2392, pp. 471–485. Springer (2002)
22. Narendran, P., Otto, F.: Single versus simultaneous equational unification and equational unification for variable-permuting theories. J. Autom. Reason. **19**(1), 87–115 (1997)
23. Nieuwenhuis, R.: Decidability and complexity analysis by basic paramodulation. Inf. Comput. **147**(1), 1–21 (1998)
24. Satterfield, S.D., Erbatur, S., Marshall, A.M., Ringeissen, C.: Knowledge problems in security protocols: Going beyond subterm convergent theories. In: Gaboardi, M., van Raamsdonk, F. (eds.) 8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy. LIPIcs, vol. 260, pp. 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)
25. Schmidt-Schauß, M.: Unification in permutative equational theories is undecidable. J. Symb. Comput. **8**(4), 415–421 (1989)