The use of types in designing unification algorithms: two case studies*

Serdar Erbatur[1], Santiago Escobar[2], and Paliath Narendran[3]

[1] Universitá degli Studi di Verona, serdar.erbatur@univr.edu
[2] Universitat Politècnica de València (Spain), sescobar@dsic.upv.es
[3] University at Albany–SUNY (USA), dran@cs.albany.edu

Abstract

We discuss the use of type systems in a non-strict sense when designing unification algorithms. We first give a new (rule-based) algorithm for an equational theory which represents a property of El-Gamal signature schemes and show how a type system can be used to prove termination of the algorithm. Lastly, we reproduce a termination result for theory of partial exponentiation given earlier.

1 Introduction

In this work, we study the use of types to show termination of unification algorithms defined for equational theories. We give two nontrivial examples of such theories, which are both shown to have decidable unification problems. First, we investigate unification modulo the theory[1], denoted as $\mathcal{E}$, which consists of following axiom:

$$B(m_1, r_1) * B(m_2, r_2) \;=\; B(m_1 * m_2,\; r_1 \circledast r_2)$$

Second, we focus on the theory of partial exponentiation, denoted as $\mathcal{F}$, for which unification was shown to be decidable in [9]. There are two axioms in $\mathcal{F}$:

$$\begin{array}{rcl} exp(g(X), Y) & = & g(X \circledast Y) \\ exp(X * Y, Z) & = & exp(X, Z) * exp(Y, Z) \end{array}$$

We introduce type systems for both equational theories and show termination of inference-rule-based algorithms by proving some results on equivalence classes of terms that have a special type. Namely, for $\mathcal{E}$ we define the following type system:

$$B : \alpha \times \gamma \to \alpha,\;\; * : \alpha \times \alpha \to \alpha,\;\; \circledast : \gamma \times \gamma \to \gamma$$

For $\mathcal{F}$, we use a different type system:

$$exp : \alpha \times \gamma \to \alpha,\;\; g : \pi \to \alpha,\;\; * : \alpha \times \alpha \to \alpha,\;\; \circledast : \pi \times \gamma \to \pi$$

Termination in both cases is based on giving rules for detecting infinite applications of inference rules. We construct such rules by defining a relation among the equivalence classes with a certain type such that this relation becomes cyclic if and only if applications of inference rules can go on forever.

*Paliath Narendran and Serdar Erbatur were partially supported by the NSF grant CNS-09-05286, while Serdar was in the Department of Computer Science of the University at Albany. Santiago Escobar has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02, and by Generalitat Valenciana PROMETEO2011/052.

[1]An abridged version of this decidability result is presented in [4].

The type systems we introduce are not strict. In fact, we keep in mind that $\mathcal{E}$ and $\mathcal{F}$ are not typed theories. We consider types as *attributes* that are attached to variables while running the algorithm. For instance, let us consider $\mathcal{E}$ and its associated type system. A term such as $B(X, X)$ would be problematic for strict typing but types can be understood as attributes in this case. This also explains how types are assigned to existing variables. As an example, the equation $U_1 =^? V * W$ and $U_2 =^? V \circledast W$ are typed properly in this discipline: U_1 with α, U_2 with γ and both V and W with α and γ, respectively. Similar ideas apply to $\mathcal{F}$ and its type system. Assignment of types to variables is based on the same idea for any given theory, however, details differ since each theory has a different set of types — see Section 3 and 4.

The outline is as follows. We give some basic definitions in Section 2. We follow the standard notation in the literature, see [2] for more details. Unification modulo $\mathcal{E}$ is explained in Section 3. In Section 4, use of types in unification modulo $\mathcal{F}$ is described.

2 Preliminaries and Notation

Let Σ be a signature, i.e., a set of function symbols and E be a set of identities based on Σ. We call E an equational theory and define *unification modulo E* or E-unification as follows. Let $\mathcal{S} = \{s_1 =^?_E t_1, \ldots, s_m =^?_E t_m\}$ be a set of equations where s_i, t_i are terms based on Σ and variables. Then $\mathcal{S}$ is called an E-unification problem. An *E-unifier* for $\mathcal{S}$ is a substitution σ such that $\sigma(s_i) =_E \sigma(t_i)$ for all $1 \leq i \leq m$. That is, *equality modulo E*, $=_E$, in $\mathcal{S}$ is satisfied if we apply σ to every equation. We use $Var(\mathcal{S})$ to denote the set of variables that occur in $\mathcal{S}$.

A set of equations S is said to be in *standard form* over a signature F if and only if every equation in S is of the form $X =^? t$ where X is a variable and t, a term over F, is one of the following: (a) a variable different from X, (b) a constant, or (c) a term of depth 1 that contains no constants. We say S is *in standard form* if and only if it is in standard form over the entire signature.

A set of equations (i.e., a unification problem) is said to be in *dag-solved-form* [8] (or *d-solved form*) if and only if they can be arranged as a list

$$x_1 =^? t_1, \ \ldots, \ x_n =^? t_n$$

where (a) each left-hand side x_i is a distinct variable, and (b) $\forall\, 1 \leq i \leq j \leq n$: x_i does not occur in t_j. It is not hard to see that a unification problem in *dag*-solved form has a unique most general unifier which can be obtained in a straightforward way [8]. If a set of equations $\mathcal{EQ}$ is in dag-solved form, we say that $\mathcal{EQ}$ is *solved.*

3 Unification modulo $\mathcal{E}$

In this section, we consider an axiom which is observed in El Gamal encryption. We briefly explain how a message is encrypted within the El Gamal scheme. Let p be a prime, g a generator of Z_p and x the private key obtained from the range 1 to $p - 2$. Define $h = g^x \, mod\, p$. The public key is the tuple (p, g, h). A message m is encrypted by first selecting a random integer r such that $gcd(r, p - 1) = 1$ and then computing

$$a \equiv g^r \, mod\, p, \ b \equiv m * h^r \, mod\, p$$

The ciphertext of m is the pair (a, b). Let us define $B(m, r) = m * h^r$. When using this functionality, there is an important property of B that could be taken into account. This property is unfolded as follows:

$$B(m_1, r_1) * B(m_2, r_2) \;=\; m_1 * h^{r_1} * m_2 * h^{r_2}$$

and

$$B(m_1, r_1) * B(m_2, r_2) \;=\; m_1 * m_2 * h^{r_1+r_2} \;(mod\, q)$$

where $h^{r_1+r_2}$ can be written (abstracted) as $r_1 \circledast r_2$. Therefore the equality of interest in this protocol is

$$B(m_1, r_1) * B(m_2, r_2) \;=\; B(m_1 * m_2,\; r_1 \circledast r_2)$$

We present decidability of $\mathcal{E}$-unification by constructing an algorithm along with a proof of correctness. First, we define a set of inference rules based on standard forms and observe that those rules are complete and sound similarly to the case in [1]. This is not very different from our approach in earlier papers, but the novelty is that termination of the algorithm (i.e., inference rules) is obtained by introducing a type system for function symbols of $\mathcal{E}$. Note that $\mathcal{E}$ is not defined as a typed theory. However, using types as described later in this section allows us to identify a set of equivalence classes that does not grow. Then through a series of lemmas we show how to detect infinite splitting, which is the hardest type of failure to detect since new variables are continuously introduced into an $\mathcal{E}$-unification problem. Thus, the algorithm either transforms an $\mathcal{E}$-unification problem to dag-solved form or returns failure by finding (i) a function clash, (ii) (extended) cycle induced by relations among the variables, or (iii) variables which split indefinitely.

The function symbols in B, $*$ and $\circledast$ are cancellative, i.e., if s_1, t_1, s_2, t_2 are ground terms in normal form, then $B(s_1, t_1) =_{\mathcal{E}} B(s_2, t_2)$ if and only if $s_1 =_{\mathcal{E}} s_2$ and $s_2 =_{\mathcal{E}} t_2$; similarly for '$*$'. This can be shown using the fact that $\mathcal{E}$ can be oriented either way to get a convergent rewrite system.

We now define several relations among variables:

- $U \succ_{r_*} V$ iff there is an equation $U = T * V$
- $U \succ_{l_*} V$ iff there is an equation $U = V * T$
- $U \succ_{r_\circledast} V$ iff there is an equation $U = T \circledast V$
- $U \succ_{l_\circledast} V$ iff there is an equation $U = V \circledast T$
- $U \succ_{r_B} V$ iff there is an equation $U = B(T, V)$
- $U \succ_{l_B} V$ iff there is an equation $U = B(V, T)$
- $U \succ_{\circledast} V$ iff $U \succ_{r_\circledast} V$ or $U \succ_{l_\circledast} V$
- $U \succ_{*} V$ iff $U \succ_{r_*} V$ or $U \succ_{l_*} V$
- $U \succ_{B} V$ iff $U \succ_{r_B} V$ or $U \succ_{l_B} V$

Let $\sim_{lp(*)}$ and $\sim_{lp(B)}$ be the reflexive, symmetric and transitive closures of $\succ_{l_*}$ and $\succ_{l_B}$ respectively. Also, let $\succ = \succ_{\circledast} \,\cup\, \succ_{*} \,\cup\, \succ_{B}$. The inference rules for $\mathcal{E}$-unification are as follows.

(a) *Variable Elimination*:

$$\frac{\{X =^? V\} \uplus \mathcal{EQ}}{\{X =^? V\} \cup [V/X](\mathcal{EQ})} \qquad \text{if } X \text{ occurs in } \mathcal{EQ}$$

(b) *Cancellation on B*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? B(W,T)\}}{\mathcal{EQ} \cup \{X =^? B(V,Y),\ V =^? W,\ Y =^? T\}}$$

(c) *Cancellation on '$*$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V * Y,\ X =^? W * T\}}{\mathcal{EQ} \cup \{X =^? V * Y,\ V =^? W,\ Y =^? T\}}$$

(d) *Cancellation on '$\circledast$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V \circledast Y,\ X =^? W \circledast T\}}{\mathcal{EQ} \cup \{X =^? V \circledast Y,\ V =^? W,\ Y =^? T\}}$$

(e) *Splitting*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W * Z\}}{\mathcal{EQ} \cup \{X =^? W * Z,\ V =^? V_0 * V_1,\ Y =^? Y_0 \circledast Y_1,\ W =^? B(V_0,Y_0),\ Z =^? B(V_1,Y_1)\ \}}$$

(f) *Failure Rule 1*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W \circledast T\}}{FAIL}$$

(g) *Failure Rule 2*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V * Y,\ X =^? W \circledast T\}}{FAIL}$$

(h) *Occur-Check*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } X \succ^+ X \text{ for some } X$$

The variable X in the splitting rule is called an *e-peak*. That is, an *e-peak* is a variable X such that $X \succ_{l_*} W$, $X \succ_{r_*} Z$, $X \succ_{l_B} V$ and $X \succ_{r_B} Y$ for some variables V, Y, W, Z. The rules (a) – (h) can be applied in any order but we propose the following strategy for efficiency in reaching the dag-solved form. Rules (a), (f), (g) and (h) have the highest priority, followed by (b), (c) and (d). Finally the rule (e) has the lowest priority.

Lemma 3.1. *Rules (a) – (h) are sound and complete for $\mathcal{E}$-unification.*

Proof. The result is obtained in a similar way to that of [1]. □

By Lemma 3.1 we find that Infinite Splitting is a failure case. A necessary and sufficient condition that uses this as a reason for non-unifiability will be given later in this paper. Two example cases that rule (e) applies infinitely by using a variable shared between the first argument of B and an argument of $*$ are given below. We underline those equations that are repeated after applying the inference rule.

Example 1: Infinite Splitting

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? V * Z\}}{\mathcal{EQ} \cup \{X =^? V * Z,\ \underline{V =^? V_0 * V_1},\ Y =^? Y_0 \circledast Y_1,\ \underline{V =^? B(V_0,Y_0)},\ Z =^? B(V_1,Y_1)\ \}}$$

Example 2: Infinite Splitting

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W * V\}}{\mathcal{EQ} \cup \{X =^? W * V,\ \underline{V =^? V_0 * V_1},\ Y =^? Y_0 \circledast Y_1,\ W =^? B(V_0,Y_0),\ \underline{V =^? B(V_1,Y_1)}\ \}}$$

Note that the case where a variable is shared between the second arguments of B and $*$ does not lead to infinite splitting:

Example 3:

$$\frac{\mathcal{EQ}\ \uplus\ \{X =^? B(V,Y),\ X =^? W * Y)\}}{\mathcal{EQ}\ \cup\ \{X =^? W * Y,\ V =^? V_0 * V_1,\ \underline{Y =^? Y_0 \circledast Y_1},\ W =^? B(V_0,Y_0),\ \underline{Y =^? B(V_1,Y_1)}\}}$$

This results in a Function Clash (failure rule (f)). In contrast, note that infinite splitting takes place in Examples 1 and 2 since both cases give rise to *e-peaks* repeatedly. To explain this, we first assign the set of types $\{\alpha, \gamma\}$ to arguments of function symbols as follows:

$$B : \alpha \times \gamma \to \alpha,\ * : \alpha \times \alpha \to \alpha,\ \circledast : \gamma \times \gamma \to \gamma$$

This type mechanism is not strict: in fact one may consider $\{\alpha, \gamma\}$ as a set of attributes. As mentioned earlier, a term such as $B(X, X)$ would be problematic for strict typing but it is reasonable to assume that X has both α and γ as "attributes" in this case [2]. This also explains how types are assigned to existing variables. As an example, the equation $U_1 =^? V * W$ and $U_2 =^? V \circledast W$ are typed properly in this discipline: U_1 with α, U_2 with γ and both V and W with α and γ, respectively.

We, in general, assign types to variables as follows. A variable V is assigned type α if and only if there exists a variable U such that one of $U \succ_* V$, $U \succ_{l_B} V$, $V \succ_* U$, or $V \succ_{l_B} U$ holds. Likewise, a variable V is assigned type γ if and only if there exists a variable U such that $U \succ_\circledast V$ or $U \succ_{r_B} V$ or $V \succ_\circledast U$ or $V \succ_{r_B} U$.

We can now observe that in (1) and (2) the new peaks have type α. However, the set $Var(\mathcal{S})$ for a problem $\mathcal{S}$ may get larger because of splitting. Also, if V is the representative of an equivalence class of variables with respect to a relation $R \in \{\sim_{lp(*)}, \sim_{lp(B)}\}$, i.e., $[V]_R$, then obviously all variables in $[V]_R$ have the same type as V.

As seen in rule (e), a variable T can split in two ways: either as $T = T_0 * T_1$ or as $T = T_0 \circledast T_1$. The splitting rule (e) may be applied further to new variables, and in general we may obtain a variable T_ω where $\omega \in \{0,1\}^*$ is a string of 0's and 1's. Therefore we adopt the general discipline for creating new variables as: $T_\omega = T_{\omega 0} * T_{\omega 1}$ **or** $T_\omega = T_{\omega 0} \circledast T_{\omega 1}$ [3]. Also, if $\omega = \lambda$, i.e., the empty string, then $T_\omega = T$, which implies T is an original variable. For a variable V, define $\overline{V} = \{V_\omega \mid \omega \in \{0,1\}^*\}$. Note that $\overline{V}$ may be infinitely large.

Lemma 3.2. *Let V be of type α and $|\overline{V}|$ be infinite. Then any descendant V_ω, where $\omega \in \{0,1\}^*$ and $V \succ_*^+ V_\omega$, joins an originally existing $\sim_{lp(B)}$-equivalence class which has type α.*

Proof. By induction on $|\omega|$ and definition of $\succ_*^+$ and $\sim_{lp(B)}$. $\square$

Our main observations are (i) if a variable splits, then its descendants have the same type, see Lemma 3.2 and (ii) in case of infinite splitting the new e-peaks are always α-typed. We justify (ii) later in this section. Thus (i) and (ii) allow us to effectively leave γ-typed variables out.

Definition 3.3. Let $\mathcal{V} = \{X_w \mid X$ is α-typed and $\omega \in \{0,1\}^*\}$ i.e., the set of variables which have type α in a given problem. In other words, $\mathcal{V}$ includes the original variables with type α and the new variables that α is assigned as type.

Lemma 3.4. *Let $\mathcal{S}$ be an $\mathcal{E}$-unification problem and $X \in Var(\mathcal{S})$ such that X is an e-peak. Then X has type α, i.e., $X \in \mathcal{V}$.*

[2]In an order-sorted environment, one could have a sort μ such that $\alpha < \mu$ and $\gamma < \mu$ and μ would represent the combination of both sorts, i.e., the case $\{\alpha, \gamma\}$

[3]Using the type discipline given earlier, we assume that T is α-typed in the former case and γ-typed in the latter.

Proof. Immediate. □

Let us consider grouping elements of $\mathcal{V}$ with respect to the equivalence relation $\sim_{lp(B)}$. That is, we write $\mathcal{V} = \biguplus [X]_{\sim_{lp(B)}}$ where $X \in \mathcal{V}$. Therefore, we have a set of $\sim_{lp(B)}$-equivalence classes such that the number of them remain the same even if the splitting applies infinitely.

Lemma 3.5. *The number of $\sim_{lp(B)}$-equivalence classes in $\mathcal{V}$ does not increase.*

Proof. Follows from the definition of $\mathcal{V}$. Note that any α-typed variable derived from splitting joins an $\sim_{lp(B)}$-equivalence class that already exists in $\mathcal{V}$. Hence the result. □

Let us define $\beta = \sim_{lp(B)} \circ \succ_* \circ \sim_{lp(B)}$. Then the following result gives us a way to detect infinite splitting.

Lemma 3.6. *If rule (e) applies infinitely, hence there is no solution, then there exists a β-cycle among $\sim_{lp(B)}$-equivalence classes in $\mathcal{V}$.*

Proof. By Lemmas 3.2 and 3.5 the equivalence classes remain the same and the new peak variables join existing classes. For every variable $X \in \mathcal{V}$ created by splitting there exist another variable $Y \in \mathcal{V}$ suct that $Y \succ_* X$ holds. Thus in case that splitting applies indefinitely we form arbitrarily long chain such as

$$X_{i_1} \succ_* X_{i_1} \ldots$$

But since $\sim_{lp(B)}$-equivalence classes do not increase, there are indices j and k s.t. $j < k$ and $X_{i_j} \sim_{lp(B)} X_{i_k}$. Then we are done since this will cause β to be cyclic among the $\sim_{lp(B)}$-equivalence classes in $\mathcal{V}$ after finitely many steps. □

We define an interpretation which gives a valid model for $\mathcal{E}$. If B is interpreted as projection to its first argument, i.e., left projection, then we get $m * n = m * n$ out of $\mathcal{E}$. This is useful since the unification problem is solvable only if its interpreted version is also solvable.

Lemma 3.7. *If β is cyclic, the $\mathcal{E}$-unification problem is not solvable.*

Proof. Let $\mathcal{P}$ be an $\mathcal{E}$-unification problem and β is cyclic. When B is interpreted as explained above, the variables in the same $\sim_{lp(B)}$-equivalence class will be equal to each other. If we denote the interpreted problem as $\mathcal{P}'$ then one should detect that $\succ_*$ is cyclic (Note that $\succ_*$ is what remains from β after interpretation). Thus $\mathcal{P}'$ is not solvable. This implies that $\mathcal{P}$ is not solvable. □

Therefore we can define the following failure rule which deals with infinite splitting.

(i) *Infinite Splitting*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } \beta \text{ is cyclic in } \mathcal{V}$$

Lemma 3.8. *Unification modulo $\mathcal{E}$ is decidable using rules (a)-(i) above.*

Proof. Let $\mathcal{P}$ be an $\mathcal{E}$-unification problem. If $\mathcal{P}$ is unifiable, then rules (a) - (e) will return an $\mathcal{E}$-unifier. But if $\mathcal{P}$ is not unifiable, one has to consider the possible failure. Two of them, namely extended occur check and function clash, are handled by rules (f), (g) and (h). In addition, there might occur infinite splitting. This case is detected by rule (i) which was justified by Lemmas 3.2 through 3.7. □

4 Unification modulo $\mathcal{F}$

The theory below is called *partial exponentiation*, and denoted as $\mathcal{F}$ here.

$$\begin{array}{rcl} exp(g(X), Y) & = & g(X \circledast Y) \\ exp(X * Y, Z) & = & exp(X, Z) * exp(Y, Z) \end{array}$$

The signature of $\mathcal{F}$ is $\{exp, g, \circledast, *\}$, where exp is the exponentiation operator, g stands for exponentiation over a fixed base and $\circledast$ and $*$ are the multiplication operators modulo different primes. This theory represents the interaction between exponentiation and multiplication in various cryptographic protocols.

Unification modulo this theory was shown to be decidable in [9]. Hence we merely give the inference rules from [9] here, skipping the proof of their completeness and soundness. Our main focus is on proving termination of the inference rules using a type system. A slightly different theory was proved terminating for unification in an order-sorted environment in [5]. The following relations are used to detect failure cases as in [9].

- $U \succ_b V$ iff there is an equation $U = exp(V, W)$
- $U \succ_e V$ iff there is an equation $U = exp(W, V$
- $U \succ_{r_*} V$ iff there is an equation $U = T * V$
- $U \succ_{l_*} V$ iff there is an equation $U = V * T$
- $U \succ_{r_\circledast} V$ iff there is an equation $U = T \circledast V$
- $U \succ_{l_\circledast} V$ iff there is an equation $U = V \circledast T$
- $U \succ_\circledast V$ iff $U \succ_{r_\circledast} V$ or $U \succ_{l_\circledast} V$
- $U \succ_m V$ iff $U \succ_{r_*} V$ or $U \succ_{l_*} V$
- $U \succ_g V$ iff there is an equation $U = g(V)$
- $U \succ V$ iff there is an equation $U = t$ where t is a non-variable term that contains V.

We introduce the following type system on $\mathcal{F}$;

$$exp : \alpha \times \gamma \to \alpha, \;\; g : \pi \to \alpha, \;\; * : \alpha \times \alpha \to \alpha, \;\; \circledast : \pi \times \gamma \to \pi$$

We assign types to variables as follows. A variable V is assigned type α if and only if there exists a variable U such that one of $U \succ_m V$, $U \succ_b V$, $V \succ_m U$, $V \succ_b U$ or $V \succ_g U$ holds. Likewise, a variable V is assigned type γ if and only if there exists a variable U such that $U \succ_{r_\circledast} V$ or $U \succ_e V$. A variable X is assigned type π if and only if there exists a variable W such that one of $W \succ_{l_\circledast} X$ or $X \succ_{l_\circledast} W$ or $W \succ_g X$ holds.

The algorithm in [9] consists of the following inference rules along with *failure rules* which we do not list here:

(a) *Variable Elimination*:

$$\frac{\{X =^? V\} \uplus \mathcal{EQ}}{\{X =^? V\} \cup [V/X](\mathcal{EQ})} \qquad \text{if } X \text{ occurs in } \mathcal{EQ}$$

(b) *Cancellation on 'exp'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? exp(V,Y),\ X =^? exp(W,T)\}}{\mathcal{EQ} \cup \{X =^? exp(V,Y),\ V =^? W,\ Y =^? T\}}$$

(c) *Cancellation on 'g'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? g(V),\ X =^? g(W)\}}{\mathcal{EQ} \cup \{X =^? g(V),\ V =^? W\}}$$

(d) *Cancellation on '$*$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V * Y,\ X =^? W * T\}}{\mathcal{EQ} \cup \{X =^? V * Y,\ V =^? W,\ Y =^? T\}}$$

(e) *Cancellation on '$\circledast$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V \circledast Y,\ X =^? W \circledast T\}}{\mathcal{EQ} \cup \{X =^? V \circledast Y,\ V =^? W,\ Y =^? T\}}$$

(f) *Splitting 1*:

$$\frac{\mathcal{EQ} \uplus \{X =^? exp(V,Y),\ X =^? g(W)\}}{\mathcal{EQ} \cup \{X =^? g(W),\ V =^? g(V'),\ W =^? V' \circledast Y\}}$$

(g) *Splitting 2*:

$$\frac{\mathcal{EQ} \uplus \{X =^? exp(V,Y),\ X =^? W * Z\}}{\mathcal{EQ} \cup \{X =^? W * Z,\ V =^? V_0 * V_1,\ W =^? exp(V_0,Y),\ Z =^? exp(V_1,Y)\ \}}$$

These rules are applied according to the following priorities: first, rule (a) is applied eagerly, then rules (b) - (e), finally one of rules (f) or (g).

4.1 Proof of Termination

Termination of the algorithm relies on showing how to detect infinite splitting. Note that all rules except (g) can be shown to terminate by defining a measure which decreases each time they are applied. Therefore we construct a necessary and sufficient failure rule by making use of types introduced for $\mathcal{F}$.

For our purpose, we recall the definition of a relation and its corresponding equivalence relation in [9]. We define the equivalence relation $\sim$ to be the reflexive, symmetric, transitive closure of $\succ_b$. Note that rule (f) increases the number of $\sim$-equivalence classes, while rule (g) does not. Also, rule (g) may be applied infinitely many times, for instance in the case $\{X =^? exp(V, Y),\ X =^? V * Z\}$.

As in the previous section, we define the set $\mathcal{V} = \{X_w \mid X \text{ is } \alpha\text{-typed and } \omega \in \{0,1\}^*\}$, i.e., variables which have type α in a given problem. Also, we consider $\mathcal{V}$ as a union of $\sim$-equivalence classes, that is, we write $\mathcal{V} = \biguplus [X]_{\sim_{lp(B)}}$ where $X \in \mathcal{V}$.

Lemma 4.1. *The number of $\sim$ equivalence classes in $\mathcal{V}$ never increases.*

Proof. Note that rule (f) introduces a new $\sim$-equivalence class, namely the class of V'. However, the type of V' is π according to the type system defined above. Thus $V' \notin \mathcal{V}$. Rule (g) does not introduce new $\sim$-equivalence classes, since we have $W \sim V_0$ and $Z \sim V_1$ and V_0 and V_1 are α-typed. Therefore we are done. □

Let us define $\beta\ = \sim\ \circ \succ_m \circ \sim$. Then we obtain the following two results similar to Lemmas 3.6 and 3.7. We skip the proofs.

Lemma 4.2. *If rule (g) applies infinitely (hence there is no solution), then β is cyclic among the $\sim$-equivalence classes in $\mathcal{V}$.*

Lemma 4.3. *If β is cyclic, the $\mathcal{F}$-unification problem is not solvable.*

Hence we can define the following failure rule that detects infinite application of rule (g).

(h) *Infinite Splitting*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } \beta \text{ is cyclic in } \mathcal{V}$$

Thus the following result is immediate:

Lemma 4.4. *The algorithm terminates.*

5 Conclusion

We have shown two case studies for the use of types in designing unification algorithms for equational theories $\mathcal{E}$ and $\mathcal{F}$, which are observed as properties cryptographic protocols. The theory $\mathcal{E}$ satisfies a property of El Gamal encryption and $\mathcal{F}$ is the theory of partial exponentiation that is observed in various protocols (see [5] for example).

Although $\mathcal{E}$ and $\mathcal{F}$ are not typed, it turns out that the type systems defined for each are useful in developing inference rules that are necessary and sufficient to detect infinite splitting. This leads us to a new method to prove termination of designed unification algorithms. Future work involves extending this method to a larger class of equational theories.

References

[1] S. Anantharaman, S. Erbatur, C. Lynch, P. Narendran, M. Rusinowitch. "Unification modulo Synchronous Distributivity". Technical Report SUNYA-CS-12-01, Dept. of Computer Science, University at Albany—SUNY. Available at `www.cs.albany.edu/~ncstrl/treports/Data/README.html` (An abridged version was presented at *IJCAR 2012*: Lecture Notes in Computer Science 7364, 14–29.)

[2] F. Baader, W. Snyder. "Unification Theory". *Handbook of Automated Reasoning*, pp. 440–526, Elsevier Sc. Publishers B.V., 2001.

[3] S. Erbatur, C. Lynch, P. Narendran. "Unification in Blind Signatures". FTP 2011 – International Workshop on First-Order Theorem Proving, Bern, Switzerland. Available at `www.cs.albany.edu/~se/blindsig_ftp2011.pdf`

[4] S. Erbatur, S. Escobar, P. Narendran. "Unification modulo a property of the El Gamal Encryption Scheme". UNIF 2012 – The 26th Workshop on Unification, Manchester, UK, 2012.

[5] S. Escobar, J. Hendrix, C. Meadows, J. Meseguer. "Diffie-Hellman Cryptographic Reasoning in the Maude-NRL Protocol Analyzer". In: *Informal Proceedings of 2nd International Workshop on Security and Rewriting Techniques (SecReT 2007)* , 2007.

[6] S. Escobar, C. Meadows, J. Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures* LNCS 5705, pages 1–50.

[7] S. Escobar, C. Meadows, D. Kapur, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, R. Sasse. "Protocol analysis in Maude-NPA using unification modulo homomorphic encryption". In: *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 65–76.

[8] J.-P. Jouannaud, C. Kirchner. “Solving equations in abstract algebras: a rule-based survey of unification.” In: *Computational Logic: Essays in Honor of Alan Robinson*, pp. 360–394, MIT Press, Boston (1991).

[9] D. Kapur, A. M. Marshall, P. Narendran. ‘Unification modulo a partial theory of exponentiation” *Proceedings 24th International Workshop on Unification* , EPTCS (42) pp. 12–23 (2010).

[10] C. Meadows. “Formal Verification of Cryptographic Protocols: A Survey” *ASIACRYPT*, pp. 135–150 (1994).