College of Computing and Information
Computer Science Department

Unification modulo Cipher Block Chaining

Siva Anantharaman[1]
Christopher Bouchard
Paliath Narendran
Michael Rusinowitch[2]

[1]LIFO - Université d'Orléans (France), e-mail: siva@univ-orleans.fr
[2]Loria-INRIA Lorraine, Nancy (France), e-mail: rusi@loria.fr

Abstract

We model chaining in terms of a simple, convergent, rewrite system over a signature with two disjoint sorts: *list* and *element*. By interpreting a particular symbol of this signature suitably, the rewrite system can model several practical situations of interest. An inference procedure is presented for deciding the unification problem modulo this rewrite system. The procedure is modular in the following sense: any given problem is handled by a system of 'list-inferences', and the set of equations thus derived between the element-terms of the problem is then handed over to any ('black-box') procedure which is complete for solving these element-equations. An example of application of this unification procedure is given, as attack detection on a Needham-Schroeder-like protocol employing the CBC encryption mode.

Keywords: Equational unification, Block chaining, Protocol.

1 Introduction

The Chaining technique is applicable in many situations. A simple case is e.g., when we want to calculate the partial sums (resp. products) of a (not necessarily bounded) list of integers, with a given 'base' integer; such a list of partial sums (resp. products) can be calculated, incrementally, with the help of the following set of equations:

$$bc(nil,\, z) = nil, \qquad bc(cons(x, Y),\, z) = cons(h(x, z),\, bc(Y,\, h(x, z)))$$

where nil is the empty list, z is the given base integer, x is an integer variable, and Y is the the given list of integers. The partial sums (resp. products) are returned as a list, by evaluating the function bc, when $h(x, z)$ is interpreted as the sum (resp. product) of x with the given base integer z.

A more sophisticated example is the Cipher Block Chaining encryption mode (CBC, in short), employed in cryptography. This mode uses the AC-operator exclusive-or (XOR) for 'chaining the ciphers across the message blocks'; here is how this is done: let $\oplus$ stand for XOR, and $m = m_1 \,.\, m_2$ be a message decomposed as a concatenation of two message blocks m_1, m_2, of equal size; then the encryption $e(m, x)$ of m with x as key will be given by $e(m, x) = e(m_1, x) \,.\, e(m_2 \oplus e(m_1, x), x)$ (cf. e.g., [12]). The above set of equations also models the CBC encryption mode: the function $h(x, y)$ will stand in this case for the encryption $e(x \oplus y, k)$ of the message-term x XOR-ed with the initialization vector y, using the public key k of the recipient of the message. Actually, our interest in the equational theory defined by the above two equations was motivated by the possibility of such a modeling for Cipher Block Chaining, and the fact that rewrite as well as unification techniques are often employable, with success, for the formal analysis of cryptographic protocols (cf. e.g., [1, 4, 5, 6, 7], and also the concluding section).

This paper is organized as follows. In Section 2 we introduce our notation and the basic notions used in the sequel; we shall observe, in particular, that the two equations above can be turned into rewrite rules and form a convergent rewrite system over a 2-sorted signature: *lists* and *elements*. Our concern in Section 3 is the unification problem modulo this rewrite system; we present a 2-level inference system (corresponding, in a way, to the 2 sorts of the signature) for solving this problem. Although our main aim is to develop on the unification problem under the assumption that h is an interpreted function symbol (as in the two situations illustrated above), for the sake of completeness we shall also consider the case where h is a free uninterpreted symbol. The soundness and completeness of our inference procedure are established in Section 4. We shall see that while the complexity of the unification problem is polynomial over the size of the problem when h is uninterpreted, it turns out to be NP-complete when h is interpreted so that the rewrite system models CBC encryption.

2 Notation and Preliminaries

We consider a ranked signature Σ, with two *disjoint* sorts: τ_e and τ_l, consisting of binary functions *bc, cons,* h, and a constant nil, and typed as follows:

$$bc :\ \tau_l \times \tau_e \to \tau_l \quad , \quad cons :\ \tau_e \times \tau_l \to \tau_l \quad , \quad h :\ \tau_e \times \tau_e \to \tau_e \quad , \quad nil : \tau_l.$$

We also assume given a set $\mathcal{X}$ of countably many variables; the objects of our study are the (well-typed) terms of the algebra $\mathcal{T}(\Sigma, \mathcal{X})$; terms of the type τ_e will be referred to as *elements*;

and those of the type τ_l as *lists.* For better readability, the set of variables $\mathcal{X}$ will be divided into two subsets: those to which 'lists' can get assigned will be denoted with upper-case letters as: $X, Y, Z, U, V, W, \ldots$, with possible suffixes or primes; these will be said to be variables of type τ_l; variables to which 'elements' can get assigned will be denoted with lower-case letters, as: $x, y, z, u, v, w, \ldots$, with possible suffixes or primes; these will be said to be variables of type τ_e. The theory we shall be studying in this paper is defined by the two axioms (equations) already mentioned in the Introduction:

$$\begin{array}{rcl} bc(nil,\, z) & = & nil \\ bc(cons(x, Y),\, z) & = & cons(h(x, z),\, bc(Y,\, h(x, z))) \end{array}$$

It is easy to see that these axioms can both be oriented left-to-right under a suitable *lexicographic path ordering (lpo)* (cf. e.g., [8]), and that they form then a convergent – i.e., confluent and terminating – 2-sorted rewrite system. The (sorted) equational theory defined by the two axioms above will be denoted in the sequel as $\mathcal{BC}$, and referred to as 'block chaining'.

In the case where h is a free uninterpreted symbol, h will obviously be fully cancellative in the sense that for any terms s_1, t_1, s_2, t_2, we have: $h(s_1, t_1) =_{\mathcal{BC}} h(s_2, t_2)$ if and only if $s_1 =_{\mathcal{BC}} s_2$ and $t_1 =_{\mathcal{BC}} t_2$. But when h is interpreted, e.g. as for CBC, this is no longer true; in that case, h will only be *semi-cancellative* in the following sense: for any terms s_1, s_2, t, we have: $h(s_1, t) =_{\mathcal{BC}} h(s_2, t)$ if and only if $s_1 =_{\mathcal{BC}} s_2$ and $h(t, s_1) =_{\mathcal{BC}} h(t, s_2)$ if and only if $s_1 =_{\mathcal{BC}} s_2$. *In the sequel, we shall always assume the symbol h to be semi-cancellative.*

Our concern in this work is the equational unification problem modulo $\mathcal{BC}$. We assume without loss of generality (wlog) that any given $\mathcal{BC}$-unification problem $\mathcal{P}$ is in a *standard form,* i.e., $\mathcal{P}$ is given as a set of equations $\mathcal{EQ}$, each having one of the following forms:

$$\begin{array}{c} U =^? V,\ U =^? bc(V, y),\ U =^? cons(v, W),\ U =^? nil, \\ u =^? v,\ \ v =^? h(w, x),\ u =^? const \end{array}$$

where *const* stands for any ground constant of sort τ_e. The first four kinds of equations – the ones with a list variable on the left-hand side – are called *list equations,* and the rest (those which have an element variable on the left-hand side) are called *element equations.* For any problem $\mathcal{P}$ in standard form, $\mathcal{L}(\mathcal{P})$ will denote the subset formed of its list equations, and $\mathcal{E}(\mathcal{P})$ the subset of element equations. A set of element equations is said to be in *dag-solved form* (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 [11]. Such a notion is naturally extended to sets of list equations as well. In the next section we give an inference system for solving any $\mathcal{BC}$-unification problem in standard form. Its rules will transform any given problem $\mathcal{P}$ into one in d-solved form.

For better comprehension, and to facilitate presentation, in the sequel we shall denote by $\mathcal{BC}_0$ the theory defined by the rewrite system $\mathcal{BC}$ when h is a free uninterpreted symbol; and by $\mathcal{BC}_1$ the theory defined in the case where h is interpreted so that $\mathcal{BC}$ models the CBC encryption mode. Any development presented below for the theory $\mathcal{BC}$ – without further precision on h – is meant as one which will be valid for both $\mathcal{BC}_0$ and $\mathcal{BC}_1$.

3 Inference System for $\mathcal{BC}$-Unification

The inference rules have to consider two kinds of equations: the rules for the *list equations* in $\mathcal{P}$, i.e., equations whose left-hand sides (lhs) are variables of type τ_l, and the rules for the *element equations,* i.e., equations whose lhs are variables of type τ_e. Our method of solving any given unification problem will be 'modular' on these two sets of equations: The list inference rules will be shown to terminate under suitable conditions, and then all we will need to do is to solve the resulting set of element equations for h.

A few technical points need to be mentioned before we formulate our inference rules. Note first that it is not hard to see that *cons* is cancellative; by this we mean that $cons(s_1, t_1) \approx_{\mathcal{BC}} cons(s_2, t_2)$, for terms s_1, s_2, t_1, t_2, if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $t_1 \approx_{\mathcal{BC}} t_2$. On the other hand, since we assume that h is semi-cancellative we can show, by structural induction, that *bc* is also *conditionally* semi-cancellative (depending on whether its first argument is *nil* or not). A more rigorous proof of this is given in Appendix A.

Note that $U =^? bc(U, x)$ is solvable by the substitution $\{U := nil\}$; in fact this equation forces U to be *nil*, as would also the set of equations: $U =^? bc(V, y),\ V =^? bc(U, x)$. Cycles of this kind have therefore to be checked to determine whether a list variable is forced to be *nil*. This can be effectively done by defining a relation $>_{bc}$ over type τ_l variables:

$$U\ >_{bc}\ V \text{ iff there is an equation } U =^? bc(V, X).$$

If $X >_{bc}{}^{+} X$ then X has to be *nil*. A set **nonnil** of variables that cannot be *nil* for any unifying substitution is defined, recursively, as follows:

- if $U =^? cons(x, V)$ is an equation then $U \in$ **nonnil**.
- if $U =^? bc(V, x)$ is an equation and $V \in$ **nonnil** then $U \in$ **nonnil**.
- if $U =^? bc(V, x)$ is an equation and $U \in$ **nonnil** then $V \in$ **nonnil**.

We also have to account for cases where an 'occur-check' succeeds on some list variable, and the problem will be unsolvable. The simplest among such cases is when we have an equation of the form $U =^? cons(z, U)$ in $\mathcal{EQ}$. But one could have more complex unsolvable cases, where the equations involve both *cons* and *bc*; e.g., when $\mathcal{EQ}$ contains equations of the form: $U =^? cons(x, V), U =^? bc(V, y)$; the problem will be unsolvable in such a case: indeed, from the axioms of $\mathcal{BC}$, one deduces that V must be of the form $V = cons(v, V')$, for some v and V', then x must be of the form $x = h(v, y)$, and subsequently $V = bc(V', x)$, and we are back to a set of equations of the same format. We need to infer failure in such a case; for that, we define two relations on the list variables of $\mathcal{EQ}$:

- $U\ >_{cons}\ V$ iff $U = cons(z, V)$, for some z.
- $U \sim_{bc} V$ iff $U = bc(V, w)$, or $V = bc(U, w)$, for some w.

Note that $\sim_{bc}$ is the symmetric closure of the relation $>_{bc}$. The reflexive, symmetric and transitive closure of $>_{bc}$ will be denoted as $\sim^*_{bc}$.

Definition 3.1 *Let $G_l = G_l(\mathcal{P})$ be the graph whose nodes are the list variables of $\mathcal{P}$, with arcs defined as follows: From a node U on G_l there is a* directed *arc to a (not necessarily different) node V on G_l iff:*

(a) *Either* $U >_{cons} V$*: in which case the arc is labeled with* $>_{cons}$*,*

Or $U >_{bc} V$*: in which case the arc is labeled with* $>_{bc}$*.*

(b) *In the latter case,* G_l *will also have a* two-sided (undirected) *edge between* U *and* V*, which is labeled with* $\sim_{bc}$*.*

(c) *On the set of nodes on* G_l*, we define a partial relation* $\succ_l$ *by setting:* $U \succ_l V$ *iff there is a path on* G_l *from* U *to* V*, at least one arc of which has label* $>_{cons}$*.* In other words: $\succ_l = \sim_{bc}^* \circ >_{cons} \circ (\sim_{bc} \cup >_{cons})^*$.

(d) *A list variable* U *of* $\mathcal{P}$ *is said to* violate occur-check *iff* $U \succ_l U$ *on* G_l*.*

The graph $G_l = G_l(\mathcal{P})$ is called the *Propagation Graph* for $\mathcal{P}$. We formulate now the inference rules for the list equations in $\mathcal{P}$.

3.1 Inference System $\mathcal{INF}_l$ for List-Equations

(L1) *Variable Elimination*:

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

(L2) *Cancellation on cons*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? cons(x, V)\}}{\mathcal{EQ} \cup \{U =^? cons(v, W),\ v =^? x,\ W =^? V\}}$$

(L3.a) *Nil solution-1*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.b) *Nil solution-2*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ V =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.c) *Nil solution-3*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}} \qquad \text{if } V >_{bc}^* U$$

(L4.a) *Semi-Cancellation on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, x)\}}{\mathcal{EQ} \cup \{U =^? bc(W, x),\ V =^? W\}} \qquad \text{if } U \in \textbf{nonnil}$$

(L4.b) *Pushing bc below cons*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\begin{array}{l}\mathcal{EQ} \cup \quad \{V =^? cons(v, Z),\ W =^? cons(w, Z),\ U =^? cons(u, U'),\\ \qquad U' =^? bc(Z, u),\ u =^? h(v, x),\ u =^? h(w, y)\}\end{array}} \qquad \text{if } U \in \textbf{nonnil}$$

(L5) *Splitting*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(x, U_1),\ U =^? bc(V, z)\}}{\mathcal{EQ} \cup \{U =^? cons(x, U_1),\ x =^? h(y, z),\ U_1 =^? bc(V_1, x),\ V =^? cons(y, V_1)\}}$$

(L6) *Occur-Check Violation*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } U \text{ occurs in } \mathcal{EQ}, \text{ and } U \succ_l U \text{ on the graph } G_l$$

(L7) *Size Conflict*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? nil\}}{FAIL}$$

The symbol '$\uplus$' in the premises of the above inference rules stands for disjoint set union (and '$\cup$' for usual set union). The role of the Variable Elimination inference rule (L1) is to keep the propagation graph of $\mathcal{P}$ irredundant: each variable has a unique representative node on $G_l(\mathcal{P})$, up to variable equality. This rule is applied most eagerly. Rules (L2) and (L3.a)–(L3.c) come next in priority, and then (L4.a) and (L4.b). The Splitting rule (L5) is applied in the "laziest" fashion, i.e., (L5) is applied only when no other rule is applicable. The above inference rules are all "don't-care" nondeterministic.

The validity of the rule (L4.b) ('Pushing *bc* below *cons*') results from the cancellativity of *cons* and the semi-cancellativity of *bc*; note that the variables Z, U', u in the 'inferred part' of this rule (L4.b) might need to be fresh; the same is true also for the variables y, V_2 in the inferred part of the Splitting rule; but in either case, this is not obligatory if existing variables meet the requirements. We show now that such an introduction of fresh variables cannot go for ever, and that the above 7 don't-care nondeterministic rules suffice, essentially, for deciding *unifiability* modulo the axioms of $\mathcal{BC}$.

Proposition 3.2 *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem, given in standard form. The system $\mathcal{INF}_l$ of list inference rules, given above, terminates on $\mathcal{P}$ in polynomially many steps.*

Proof: The variable elimination rule (L1) removes nodes from the propagation graph, while the list inference rules (L2) through (L4.a) eliminate a (directed) outgoing arc from some node of G_l. Thus their termination is easy to check. Therefore the system of list inference rules will terminate if the splitting rule (L5) terminates and the rule (L4.b) (*Pushing bc below cons*) terminates. We show that if occur-check violation (L7) does not occur, then applications of the rule (L5) or of the rule (L4.b) cannot go on forever.

First of all, observe that though the splitting rule may introduce new variables, the number of $\sim_{bc}^*$-equivalence classes of nodes cannot increase, since the (possibly) new variable V_1 belongs to the same equivalence class as U_1 ($V_1 \sim_{bc} U_1$). Thus applying the splitting rule (L5) on a list-equation $U = bc(V, x)$ removes that equation and creates a list equation of the form $U_1 = bc(V_1, x')$ for some list variables U_1 and V_1, such that $V \sim_{bc} U >_{cons} U_1 \sim_{bc} V_1$.

Suppose now that applying the splitting rule does not terminate. Then, at some stage, the graph of the derived problem will have a sequence of nodes of the form $U_0 = U >_{cons} U_1 >_{cons} \cdots >_{cons} U_n$, such that its number of nodes n strictly exceeds the initial number of $\sim_{bc}^*$-equivalence classes – which cannot increase under splitting, as was observed above. So there must exist indices $0 \leq i < j \leq n$ such that $U_j \sim_{bc}^* U_i$; in other words, we would have $U_i \succ_l U_i$, and that would have caused the inference procedure to terminate with FAIL. We conclude therefore that applying the splitting rule must terminate.

The same kind of reasoning also works for the rule (L4.b): an application of that rule removes two list equations of the form $U =^? bc(V, x), U =^? bc(W, y)$ and creates a list equation of the form $U' = bc(Z, u)$ (plus some other *cons* and/or element equations), where $U = cons(u, U')$ so $U >_{cons} U'$. We conclude, for the same reason as above, that applying rule (L4.b) must terminate.

To show that the number of steps is polynomial on the input problem size, note first that the number of nodes and edges on the Propagation Graph of $\mathcal{P}$ is polynomial in the size of $\mathcal{P}$, and that number decreases under all the inferences other than (L4.b) and (L5). But these latter rules do not increase the number of $\sim_{bc}$-edges. Let the *level* of a $\sim_{bc}$-edge be the length of the longest *cons*-path to that edge. (L4.b) and (L5) remove one $\sim_{bc}$-edge and add a new one at a greater level. Since, the length of any *cons*-chain is polynomially bounded, (L4.b) and (L5) can only be applied a polynomial number of times. □

A set of equations will be said to be *L-reduced* if none of the above inference rules (L1) through (L7) is applicable.

Unification modulo $\mathcal{BC}$: The rules (L1) through (L7) are not enough to show the existence of a unifier modulo $\mathcal{BC}$. The subset of element equations, $\mathcal{E}(\mathcal{P})$, may not be solvable; for example, the presence of an element equation of the form $\{x =^? h(x, z)\}$ should lead to failure. However, we have the following:

Proposition 3.3 *If $\mathcal{L}(\mathcal{P})$ is in L-reduced form, then $\mathcal{P}$ is unifiable modulo $\mathcal{BC}$ if and only if the set $\mathcal{E}(\mathcal{P})$ of its element equations is solvable.*

Proof: If $\mathcal{L}(\mathcal{P})$ is L-reduced, then setting every list variable that is not in **nonnil** to *nil* will lead to *a unifier* for $\mathcal{L}(\mathcal{P})$, modulo $\mathcal{BC}$, provided $\mathcal{E}(\mathcal{P})$ is solvable. □

Recall that $\mathcal{BC}_0$ is the theory defined by $\mathcal{BC}$ when h is uninterpreted.

Proposition 3.4 *Let $\mathcal{P}$ be any $\mathcal{BC}_0$-unification problem, given in standard form. Unifiability of $\mathcal{P}$ modulo $\mathcal{BC}_0$ is decidable in polynomial time (wrt the size of $\mathcal{P}$).*

Proof: If the inferences of $\mathcal{INF}_l$ applied to $\mathcal{P}$ lead to failure, then $\mathcal{P}$ is not unifiable modulo $\mathcal{BC}$; so assume that this is not the case, and replace $\mathcal{P}$ by an equivalent problem which is L-reduced, deduced in polynomially many steps by Proposition 3.2. By Proposition 3.3, the unifiability modulo $\mathcal{BC}$ of such a $\mathcal{P}$ amounts to checking if the set $\mathcal{E}(\mathcal{P})$ of its element equations is solvable. We are in the case where h is uninterpreted, so to solve $\mathcal{E}(\mathcal{P})$ we apply the rules for standard unification, and check for their termination without failure; this can be done in polynomial time; cf. e.g., [3]. (In this case, h is fully cancellative.) □

It can be seen that, while termination of the above inference rules guarantees the *existence* of a unifier (provided the element equations are syntactically solvable), the resulting L-reduced system may not lead directly to a unifier. For instance, the system of list equations: $\{U =^? bc(V, x),\ U =^? bc(V, y)\}$. is L-reduced system is unifiable, with two incomparable unifiers; namely:

$$\{x := y,\ U := bc(V, y)\} \text{ and } \{U := nil,\ V := nil\}$$

To get this complete set of unifiers we need two more inference rules, which are "don't-know" nondeterministic, and to be applied only to L-reduced systems:

(L8) *Nil-solution-Branch*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? bc(V,y)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L9) *Cancellation-Branch on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? bc(V,y)\}}{\mathcal{EQ} \cup \{U =^? bc(V,y),\ x =^? y\}} \qquad \text{if } U \in \mathbf{nonnil}$$

We establish now a technical result, valid whether or not h is interpreted:

Proposition 3.5 *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem in standard form, to which none of the inferences of $\mathcal{INF}_l$ is applicable. Then its set of list-equations is in d-solved form.*

Proof: Observe first that if $\mathcal{INF}_l$ is inapplicable to $\mathcal{P}$, then, on the propagation graph G_l for $\mathcal{P}$, there is *at most one outgoing directed arc* of G_l at any node U: Otherwise, suppose there are two distinct outgoing arcs at some node U on G_l; if both directed arcs bear the label $>_{cons}$, then rule (L2) of $\mathcal{INF}_l$ would apply; if both bear the label $>_{bc}$, then either (L4.a) or (L4.b) would apply; the only remaining case is where one of the outgoing arcs is labeled with $>_{cons}$ and the other has label $>_{bc}$, but then the splitting rule (L5) would apply.

Consider now any given connected component Γ of G_l. There can be no directed cycle from any node U on Γ to itself: otherwise the Occur-Check-Violation rule (L6) would have applied. It follows, from this observation and the preceding one, that there is a unique *end-node* U_0 on Γ – i.e., a node from which there is *no directed outgoing arc* –, and also that for any given node U on Γ, there is a unique well-defined directed path leading from U to that end-node U_0.

It follows easily from these, that the left-hand-side list-variables of $\mathcal{P}$ (on the different connected components of G_l) can be ordered suitably so as to satisfy the condition for $\mathcal{P}$ to be in d-solved form. □

Example 3.6 *The following $\mathcal{BC}_0$-unification problem: is in standard form:*

$$U =^? cons(x,W),\ U =^? bc(V,y),\ W =^? bc(V_3,y),\ x =^? h(z,y),\ y =^? a$$

*We apply splitting, by writing $V =^? cons(v_2,V_2)$, where v_2, V_2 are fresh; we deduce, after one application of (L2) (*Cancellation on cons*):*

$$U =^? cons(x,W),\ V =^? cons(v_2,V_2),\ W =^? bc(V_2,y),\ W =^? bc(V_3,y)$$
$$x =^? h(v_2,y),\ x =^? h(z,y),\ y =^? a$$

*To which, rule (L4.a) (*Semi-Cancellation on *bc) and the (semi-)cancellativity of h apply; we finally get*

$$U =^? cons(x,W),\ V =^? cons(z,V_3),\ W =^? bc(V_3,y),\ x =^? h(z,y),\ y =^? a$$

as solution, where z and V_3 can be taken as arbitrary. Note that this final set of equations is in d-solved form; and also that its the propagation graph is connected, with V_3 as the end-node.

We now turn our attention to solving a unification problem modulo the theory $\mathcal{BC}$. When h is uninterpreted, we saw above that this unification is decidable in polynomial time; but in the case where h is interpreted so that $\mathcal{BC}$ models CBC, we shall see that unification modulo $\mathcal{BC}_1$ is NP-complete.

4 Solving a $\mathcal{BC}$-Unification problem

Let $\mathcal{P}$ be a $\mathcal{BC}$-Unification problem, given in standard form. We assume that $\mathcal{INF}_l$ has terminated without failure on $\mathcal{P}$; we saw, in the preceding section (Proposition 3.5), that $\mathcal{P}$ is then in d-solved form. We also assume that we have a sound and complete procedure for solving the element equations of $\mathcal{P}$, that we shall denote as $\mathcal{INF}_e$. For the theory $\mathcal{BC}_0$ where h is uninterpreted, we know (Proposition 3.4) that $\mathcal{INF}_e$ is standard unification, with cancellation rules for h, and failure in case of 'symbol clash'. For the theory $\mathcal{BC}_1$, where $h(x, y)$ is interpreted as $e(x \oplus y, k)$ for some fixed key k, $\mathcal{INF}_e$ will have rules for semi-cancellation on h and e, besides the rules for unification modulo XOR in some fixed procedure, that we assume given once and for all.

In all cases, we shall consider $\mathcal{INF}_e$ as a black-box that either returns most general unifiers (mgu's) for the element equations of $\mathcal{P}$, or a failure message when these are not satisfiable. Note that $\mathcal{INF}_e$ is unitary for $\mathcal{BC}_0$ and finitary for $\mathcal{BC}_1$. For any problem $\mathcal{P}$ in d-solved form, satisfiable under the theory $\mathcal{BC}_0$, there is a unique mgu, as expressed by the equations of $\mathcal{P}$ themselves (cf. also [11]), that we shall denote by $\theta_{\mathcal{P}}$. Under $\mathcal{BC}_1$ there could be more than one (but finitely many) mgu's; we shall agree to denote by $\theta_{\mathcal{P}}$ any one among them.

The entire procedure for solving any $\mathcal{BC}$-unification problem $\mathcal{P}$, given in standard form, can now be synthesized as a nondeterministic algorithm:

The Algorithm $\mathcal{A}$: Given a $\mathcal{BC}$-unification problem $\mathcal{P}$, in standard form.
G_l = Propagation graph for $\mathcal{P}$. $\mathcal{INF}_l$ = Inference procedure for $\mathcal{L}(\mathcal{P})$.
$\mathcal{INF}_e$ = (Complete) Procedure for solving the equations of $\mathcal{E}(\mathcal{P})$.

1 Compute a standard form for $\mathcal{P}$, to which the mandatory inferences of $\mathcal{INF}_l$ are no longer applicable. If this leads to failure, exit with FAIL. Otherwise, replace $\mathcal{P}$ by this standard form.

2 Apply the "don't-know" nondeterministic rules (L8)-(L9), – followed by the rules of $\mathcal{INF}_l$ as needed – until the equations no longer get modified by the inference rules (L1)-(L9). If this leads to failure, exit with FAIL.

3 Apply the procedure $\mathcal{INF}_e$ for solving the element-equations in $\mathcal{E}(\mathcal{P})$; if this leads to failure, exit with FAIL.

4 Otherwise let σ be the substitution on the variables of $\mathcal{P}$ as expressed by the resulting equations. Return σ as a solution to $\mathcal{P}$.

Proposition 4.1 *The algorithm $\mathcal{A}$ is sound and complete.*

Proof: The soundness of $\mathcal{A}$ follows from the soundness (assumed) of $\mathcal{INF}_e$ and that of $\mathcal{INF}_l$, which is easily checked: If $\mathcal{P}'$ is any problem derived from $\mathcal{P}$ by applying any of these inference rules, then any solution for $\mathcal{P}'$ corresponds to a solution for $\mathcal{P}$. The completeness of $\mathcal{A}$ follows from the completeness (assumed) of $\mathcal{INF}_e$, and the completeness of $\mathcal{INF}_l$ that we prove below:

Lemma 4.2 *If σ is a solution for a given $\mathcal{BC}$-unification problem $\mathcal{P}$ in standard form, then there is a sequence of $\mathcal{INF}_l$-inference steps that transforms $\mathcal{P}$ into a problem $\mathcal{P}'$ in d-solved form such that σ is an instance of $\theta_{\mathcal{P}'}$ (modulo $\mathcal{BC}$).*

Proof-sketch. The proof is by case analysis. We may assume, without loss of generality, that $\mathcal{P}$ is L-reduced (i.e., the mandatory inferences of $\mathcal{INF}_l$ have all been applied). If $\mathcal{P}$ is already in d-solved form then we are done since $\sigma \preceq_{\mathcal{BC}} \theta_{\mathcal{P}}$, for some mgu $\theta_{\mathcal{P}}$. If $\mathcal{P}$ is not in d-solved form, then we have to consider several cases, depending on the possible inference branches. We will just illustrate one such case. Suppose there are two equations $U =^? bc(Z, v)$ and $U =^? bc(Y, w)$ in $\mathcal{P}$. If $\sigma(v) =_{\mathcal{BC}} \sigma(w)$, then we must have $\sigma(Z) =_{\mathcal{BC}} \sigma(Y)$, so σ must be a solution for the problem obtained by applying the rule (L4.a). If $\sigma(v) \neq_{\mathcal{BC}} \sigma(w)$, then σ must be a solution to the problem derived under rule (L4.b). Now, we know that the inference steps always terminate, so such a reasoning can be completed into an inductive argument, to prove the lemma. □

4.1 $\mathcal{BC}_1$-Unification is NP-Complete

Recall that $\mathcal{BC}_1$ is the theory defined by $\mathcal{BC}$ when the symbol h is interpreted so that $\mathcal{BC}$ models CBC.

Proposition 4.3 *Unifiability modulo the theory $\mathcal{BC}_1$ is NP-complete.*

Proof: We deduce the NP-upper bound from the following facts:

- For any given $\mathcal{BC}$-unification problem, computing a standard form is in polynomial time, wrt the size of the problem.
- Given a standard form, the propagation graph can be constructed in polynomial time (wrt its number of variables).
- Applying (L1)-(L9) till termination takes only polynomially many steps.
- Extracting the set of element-equations from the set of equations obtained in the previous step is in P.
- Solving the element-equations with the procedure $\mathcal{INF}_e$ using unification modulo XOR is in NP.

The NP-lower bound follows from the fact that general unification modulo XOR is NP-complete [10]. □

4.2 An Illustrative Example

The following public key protocol is known to be secure ([9]):

$$A \rightarrow B : \{A, m\}_{kb}$$
$$B \rightarrow A : \{B, m\}_{ka}$$

where A, B are the participants of the protocol session, m is a message that they intend secret for others, and kb (resp. ka) is the public key of B (resp. A).

However, if the CBC encryption mode is assumed and the message blocks are all of the same size, then this protocol becomes insecure, here is why. Let $e_Z(x)$ stand for the encryption $e(x, kz)$ with the public key kz of any principal Z. What A sends to B is $cons(A, cons(m, nil))$, or $[A, m]$ in ML-notation, encrypted using B's public-key. Under the CBC encryption mode, this will sent

as:

$$A \to B : [\, e_B(A \oplus v),\ e_B(m \oplus e_B(A \oplus v))\,].$$

Here $\oplus$ stands for XOR and v is the initialization vector that A and B have agreed upon. But then, some other agent I, entitled to open a session with B with initialization vector w, can get hold of the first block (namely: $e_B(A \oplus v)$) as well as the second block of what A sent to B, namely $e_B(m, e_B(A \oplus v))$; (s)he can then send the following as a 'bona fide message' to B:

$$I \to B : [\, e_B(I \oplus w),\ e_B(m \oplus e_B(A \oplus v))\,];$$

upon which B will send back to I the following:

$$B \to I : [\, e_I(B \oplus w),\ e_I(\, m \oplus e_B(A \oplus v) \oplus e_B(I \oplus w) \oplus e_I(B \oplus w)\,)\,].$$

It is clear then that the intruder I can get hold of the message m which was intended to remain secret for him/her.

Example 4.4 *The above attack (which exploits the properties of XOR: $x \oplus x = 0,\ x \oplus 0 = x$) can be reconstructed by solving a certain $\mathcal{BC}_1$-unification problem. We assume that the names A, B, I, as well as the initialization vector w, are constants. The message m and the initialization vector v, that A and B have agreed upon, are constants intended to be secret for I. We shall interpret the function symbol h of $\mathcal{BC}$ in terms of encryption with the public key of B: i.e., $h(x, y)$ is $e_B(x \oplus y)$. Due to our CBC-assumption, the ground terms $h(A, v),\ h(m, h(A, v))$ both become 'public' (i.e., accessible to I). We shall agree that a ground constant d stands for the public term $h(m, h(A, v))$.*

The above attack then amounts to saying that the intruder I can send the following term, as a message to B:

$$cons(\, h(I, w),\, [d]\,) = cons(\, h(I, w),\, cons(\, h(m, h(A, v)),\, nil\,)\,);$$

which will be considered a legitimate message by B (as mentioned earlier).

The possibility for I to construct the attack along this scheme can then be deduced by solving the following $\mathcal{BC}_1$-unification problem:

$$bc(X, h(I, w)) =^? cons(\, h(m, h(A, v)),\, nil\,)$$

*under the condition that terms in the solution are all 'public'. After transforming into standard form, we apply rule (L5) ('*Splitting'*) and write: $X =^? cons(z, Y)$, where z and Y are fresh variables. By applying rule (L2) ('*Cancellation on *cons*'*) we deduce $Y := nil$, but we still have to show that following problem:*

$$h(z, h(I, w)) =^? h(m, h(A, v))$$

is solvable for z, with public terms; using the properties of XOR we get the solution

$$z := h(\, h(m, h(A, v)),\, h(I, w)\,),$$

i.e., $z := h(\, d, h(I, w)\,)$. The solution thus derived for X is $X := [h(\, d, h(I, w)\,)]$. □

Remark: It might be of interest to note that the above reasonings do not go through if the names-tamps form the second block *in the messages sent; the protocol could be secure, in such a case, even under CBC.*

5 Conclusion

We have addressed the unification problem modulo a convergent 2-sorted rewrite system $\mathcal{BC}$, which can model, in particular, the CBC encryption mode of cryptography, by interpreting suitably the function h in $\mathcal{BC}$. A procedure has been given for deciding unification modulo $\mathcal{BC}$, which has been shown to be sound and complete when h is either uninterpreted, or interpreted in such a manner. In the uninterpreted case, the procedure is a combination of the inference procedure $\mathcal{INF}_l$ presented in this paper, with standard unification; in the case where h is interpreted as mentioned above, our unification procedure is a combination of $\mathcal{INF}_l$ with any complete procedure for deciding unification modulo the associative-commutative theory for XOR.

Although we have given an example of attack detection using our unification procedure on a cryptographic protocol employing CBC encryption, for the formal analysis of cryptographic protocols, unification needs to generalized as a procedure for solving deduction constraints [13] or, equivalently, as a cap unification procedure [2]. Building such an extension is part of our projected future work.

References

[1] S. Anantharaman, P. Narendran, M. Rusinowitch. "Intruders with Caps". In *Proc. of the Int. Conference RTA'07*, LNCS 4533, pp. 20–35, Springer-Verlag, June 2007.

[2] S. Anantharaman, H. Lin, C. Lynch, P. Narendran, M. Rusinowitch. "Cap Unification: Application to Protocol Security modulo Homomorphic Encryption". In *Proc. of the 5th ACM Symp. on Information, Computer and Communications Security*, ASIACCS'10, pp. 192–203, April 2010.

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

[4] M. Baudet. "Deciding security of protocols against off-line guessing attacks". In *Proc. of the 12th ACM Conf. on Computer and Comm. Security*, CCS'05, pp. 16–25, November 2005.

[5] H. Comon-Lundh, R. Treinen. "Easy Intruder Deductions." Verification: Theory and Practice In *LNCS* 2772, pp. 225–242, Springer-Verlag, February 2004.

[6] H. Comon-Lundh, V. Shmatikov. "Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive-Or." In *Proc. of the Logic In Computer Science Conference, LICS'03,* pp. 271–280, 2003.

[7] V. Cortier, S. Delaune, P. Lafourcade. "A Survey of Algebraic Properties Used in Cryptographic Protocols". *Journal of Computer Security* 14(1): 1–43, 2006.

[8] N. Dershowitz. "Termination of Rewriting." Journal of Symbolic Computation 3(1/2): 69–116 (1987).

[9] D. Dolev, S. Even, R. Karp, "On the Security of Ping-Pong Protocols". Information and Control, 55:57-68 (1982)

[10] Q. Guo, P. Narendran, D.A. Wolfram. "Unification and Matching Modulo Nilpotence." In *Proc. of the 13th Int. Conf. on Automated Deduction,* (CADE-13), LNCS 1104, pp. 261–274, Springer-Verlag, August 1996.

[11] J.-P. Jouannaud, and C. Kirchner. "Solving Equations in Abstract Algebras: a Rule-Based Survey of Unification." In *Computational Logic: Essays in Honor of Alan Robinson,* 360–394, MIT Press, Boston (1991).

[12] S. Kremer, M. D. Ryan. "Analysing the Vulnerability of Protocols to Produce Known-Pair and Chosen-Text Attacks". In *Proc. of the 2nd Int. Workshop on Security Issues in Coordination Models, Languages, and Systems (SecCo 2004)*, ENTCS, Vol 128, Issue 5, pp. 87–104, 2005.

[13] J. Millen, V. Shmatikov. "Constraint Solving for Bounded-Process Cryptographic Protocol Analysis." In *Proc. of the 8th ACM Conf. on Computer and Comm. Security* pp. 166–175, 2001.

A Appendix: Proof of Semi-Cancellativity of bc

Lemma A.1 *For all terms t_1, t_2, t_3,*

$$bc(t_1, t_3) \approx_{\mathcal{BC}} bc(t_2, t_3)$$

if and only if $t_1 \approx_{\mathcal{BC}} t_2$.

Proof. If $t_1 \approx_{\mathcal{BC}} t_3 \approx_{\mathcal{BC}} nil$, then this equation is trivially true. Similarly, if t_1 and t_3 equal non-nil constants, then bc must cancel and $t_1 \approx_{\mathcal{BC}} t_3$.

Suppose, then, that t_1 and t_3 are not constants. Then $t_1 \approx_{\mathcal{BC}} cons(u_1, t_1')$ and $t_3 \approx_{\mathcal{BC}} cons(u_2, t_3')$, for terms u_1, u_2, t_1', and t_3'. After substituting back into the original equation and applying the second axiom, we get

$$cons(h(u_1, t_2), bc(t_1', h(u_1, t_2))) \approx_{\mathcal{BC}} cons(h(u_2, t_2), bc(t_3', h(u_2, t_2)))$$

Since *cons* is cancellative, we get two equations

$$h(u_1, t_2) \approx_{\mathcal{BC}} h(u_2, t_2), bc(t_1', h(u_1, t_2)) \approx_{\mathcal{BC}} bc(t_3', h(u_2, t_2))$$

And since h is semi-cancellative,

$$u_1 \approx_{\mathcal{BC}} u_2, bc(t_1', h(u_1, t_2)) \approx_{\mathcal{BC}} bc(t_3', h(u_1, t_2))$$

Since we can assume there are no cycles in our set of equations, our system is well-founded. Therefore, by structural induction, $bc(t_1, t_2) \approx_{\mathcal{BC}} bc(t_3, t_2)$ if and only if $t_1 \approx_{\mathcal{BC}} t_3$. □

Lemma A.2 *For all terms t_1, t_2, t_3,*

$$bc(t_1, t_2) \approx_{\mathcal{BC}} bc(t_1, t_3)$$

if and only if $t_1 \approx_{\mathcal{BC}} nil$ or $t_2 \approx_{\mathcal{BC}} t_3$.

Proof. If $t_1 \approx_{\mathcal{BC}} nil$, then by the first axiom, we get $nil \approx_{\mathcal{BC}} nil$, which is trivially true. If t_1 is non-nil and equal to a constant, then bc must cancel and $t_2 \approx_{\mathcal{BC}} t_3$.

If t_1 is not equal to a constant, then $t_1 \approx_{\mathcal{BC}} cons(u, t_1')$, for some terms u and t_1'. Substituting back into the original equation and applying the second axiom,

$$cons(h(u, t_2), bc(t_1', h(u, t_2))) \approx_{\mathcal{BC}} cons(h(u, t_3), bc(t_1', h(u, t_3)))$$

Cancelling on *cons* gives

$$h(u, t_2) \approx_{\mathcal{BC}} h(u, t_3), bc(t_1', h(u, t_2)) \approx_{\mathcal{BC}} bc(W', h(u, t_3))$$

And since h is semicancellative,

$$t_2 \approx_{\mathcal{BC}} t_3, bc(t_1', h(u, t_2)) \approx_{\mathcal{BC}} bc(t_1', h(u, t_2))$$

Therefore, $bc(t_1, t_2) \approx_{\mathcal{BC}} bc(t_1, t_3)$ if and only if either $t_1 \approx_{\mathcal{BC}} nil$ or $t_2 \approx_{\mathcal{BC}} t_3$. □

Lemma A.3 *For all terms t_1, t_2, t_3, t_4, u_1, u_2, if*

$$bc(cons(u_1, t_1), t_2) \approx_{\mathcal{BC}} bc(cons(u_2, t_3), t_4)$$

then $h(u_1, t_2) \approx_{\mathcal{BC}} h(u_2, t_4)$ and $t_1 \approx_{\mathcal{BC}} t_3$.

Proof. By applying the second axiom, we get

$$cons(h(u_1, t_2), bc(t_1, h(u_1, t_2))) \approx_{\mathcal{BC}} cons(h(u_2, t_4), bc(t_3, h(u_2, t_4)))$$

Cancelling on *cons* gives

$$h(u_1, t_2) \approx_{\mathcal{BC}} h(u_2, t_4), \quad bc(t_1, h(u_1, t_2)) \approx_{\mathcal{BC}} bc(t_3, h(u_2, t_4))$$

Notice that, by Lemma A.1, this implies that $t_1 \approx_{\mathcal{BC}} t_3$. □