Abstraction and Resolution Modulo AC: How to Verify Diffie-Hellman-like Protocols Automatically ⋆

Jean Goubault-Larrecq [a,*] Muriel Roger [b,1]
Kumar Neeraj Verma [c,1]

[a] *LSV/CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan*
61, av. du président-Wilson, 94235 Cachan Cedex, France

[b] *DRT/LIST/DTSI/SLA/LSL*
CEA, centre de Saclay, 91191 Gif-sur-Yvette, France

[c] *Technische Universität München, Institut für Informatik / I2*
Boltzmannstraße 3, 85748 Garching, Germany

Abstract

We show how cryptographic protocols using Diffie-Hellman primitives, i.e., modular exponentiation on a fixed generator, can be encoded in Horn clauses modulo associativity and commutativity. In order to obtain a sufficient criterion of security, we design a complete (but not sound in general) resolution procedure for a class of flattened clauses modulo simple equational theories, including associativity-commutativity. We report on a practical implementation of this algorithm in the MOP modular platform for automated proving; in particular, we obtain the first fully automated proof of security of the IKA.1 initial key agreement protocol in the so-called pure eavesdropper model.

Key words: cryptographic protocols, Diffie-Hellman, resolution, clause, associativity, commutativity, completeness, abstraction, abstract interpretation, verification, key exchange

---

⋆ Partially supported by the ACI VERNAM, the RNTL projects EVA and Prouvé, the ACI Sécurité Informatique Rossignol, and the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions".

* Corresponding author. Fax: +33-1 47 40 75 21.
Email addresses: goubault@lsv.ens-cachan.fr (Jean Goubault-Larrecq), muriel.roger@cea.fr (Muriel Roger), verma@in.tum.de (Kumar Neeraj Verma).
URL: http://www.lsv.ens-cachan.fr/~goubault/ (Jean Goubault-Larrecq).
[1] Work done while PhD students at LSV, CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan on an MENRT grant.

1 Introduction

It has been said many times that verifying cryptographic protocols is important. One popular kind of model is that of Dolev and Yao (after Dolev and Yao (1983), see Comon and Shmatikov (2002) for a survey), where: the intruder can read and write on every communication channel, and in effect has full control over the network; the intruder may encrypt, decrypt, build and destruct pairs, as many times as it wishes; and, finally, cryptographic means are assumed to be *perfect*. The latter in particular means that the only way to compute the plaintext M from the ciphertext $\{M\}_K$ is to decrypt the latter using the inverse key K^{-1}. It also means that no ciphertext can be confused with any message that is not a ciphertext, and that $\{M\}_K = \{M'\}_{K'}$ implies $M = M'$ and $K = K'$. Thus, messages can be simply encoded as first-order terms, a fact which has been used by many authors; no algebraic law, except the trivial one $M = M$, holds on messages.

While this is a fine assumption if one uses encryption algorithms such as DES, RC5 or IDEA for example, modular exponentiation and elliptic curve group operations definitely obey non-trivial algebraic laws. In this paper, we are specially interested in modeling Diffie-Hellman primitives (Diffie and Hellman, 1976). While these are usually implemented through modular exponentiation, the general framework can be described by the use of one unary function e and an associative-commutative (AC), or even an Abelian group law $\oplus$ (Goubault-Larrecq and Verma, 2002; Verma, 2003a) with unit $\mathbf{0}$. For example, the original Diffie-Hellman protocol for establishing a common secret key between two principals A and B has A send the message $e(N_a)$ to B, where N_a is a nonce (implemented as a fresh, random number) created by A, then B send the message $e(N_b)$ to A, where N_b is another nonce created by B. Assuming that, once you know $e(M)$ and M', you may deduce $e(M \oplus M')$, both A and B can build the common secret key $e(N_a \oplus N_b)$. The standard implementation is by using numbers in $\mathbb{Z}/p\mathbb{Z}^*$ (p prime) for messages, letting $e(M)$ be $g^M \bmod p$ for some primitive element (generator) of $\mathbb{Z}/p\mathbb{Z}^*$, and $\oplus$ be multiplication of exponents; once you know $e(M) = g^M$ and M', you indeed know $e(M \oplus M') = g^{MM'} = (g^M)^{M'}$. Note that we may alternatively use the law of other groups, e.g., that of an elliptic curve on $\mathbb{Z}/p\mathbb{Z}$ for $\oplus$ instead.

The Diffie-Hellman primitive is used not only in the famous Diffie-Hellman protocol, but also in the so-called Diffie-Hellman ephemeral key exchange (Goldwasser and Bellare, 1999), and in some variants of El Gamal encryption and signature. In this paper, we shall take the IKA.1 group key agreement protocol (Steiner et al., 2000) as a running example of our verification techniques for protocols using Diffie-Hellman primitives — e, $\oplus$, $\mathbf{0}$ as above.

Now $\oplus$ is associative, commutative, and has $\mathbf{0}$ as a unit. For instance, $e(M \oplus$

M') should be considered as equal to $e(M' \oplus M)$, otherwise the Diffie-Hellman protocol just does not work. In IKA.1, not just commutativity, but also associativity is required for the protocol to work as intended. This calls for a modification of the Dolev-Yao model, where messages are taken to be terms modulo an equational theory: here, the theory ACU of associativity, commutativity and unit $\mathbf{0}$ of $\oplus$; and, to be more faithful to implementations based on Abelian groups such as $\mathbb{Z}/p\mathbb{Z}^*$, the richer theory of Abelian groups.

It has been shown in Goubault-Larrecq and Verma (2002) and in the third author's PhD thesis (Verma, 2003a) how special formats of first-order Horn clauses modulo ACU could be put to good use to describe faithfully the IKA.1 protocol. These formats can be seen as extensions of tree automata (Comon et al., 1997) to both so-called *two-way* tree automata and equational theories. We recap the third author's encoding of IKA.1 for a given number of principals in Section 3. In Section 5, we shall also give a sound abstraction of IKA.1 for *arbitrarily many* principals, and we shall show that it is secure in the so-called pure eavesdropper security model. (It is well-known that IKA.1 is insecure in the other models we shall consider here, see Millen and Denker (2002).)

The point is that we establish these results by an *automated* method: once the clauses describing the protocol have been written, the security of the protocol, which reduces to showing that the given set of clauses is satisfiable, is shown automatically by the MOP tool, due to the second author (Roger, 2003). In principle, the decidability results of Verma (2003a) could be used to this end, but the complexity of the corresponding algorithms are not primitive recursive; Verma (2003c)'s algorithms are more efficient (elementary) and may provide a basis for an exact verification method. In this paper, we present an approximate resolution technique that works in exponential time, and was fast enough in practice. This technique is described in Section 4, which is the technical core of the paper.

Finally, these techniques are general enough to establish the satisfiability of fairly general clause sets modulo AC, the theory of one associative-commutative symbol, and other theories. This may be of interest in other fields, too.

Outline. We first review related work in Section 2. We recapitulate the third author's encoding of the IKA.1 protocol in Section 3. This will give the reader a hint of the kind of clauses that we need to handle. Section 4 describes a conservative approximation procedure to solve such clause sets. This is based on resolution techniques (Bachmair and Ganzinger, 2001), plus a technical result that allows one to replace full subderivations by calls to a specialized oracle. Although implementing the corresponding oracle in the case of the theory AC and extensions may seem a challenge, we show how it can be implemented efficiently using counting mechanisms in Section 4.6. This has been implemented in the MOP prover, a modular platform for implementing resolution provers on general algebras, including algebras of terms modulo an equational theory (Roger, 2003). We report on practical experiments with MOP running on IKA.1 in Section 5, and conclude in Section 6.

2 Related Work

2.1 Verifying cryptographic protocols modulo algebraic laws, and Diffie-Hellman primitives

The observation that messages in the Dolev-Yao model are just first-order terms is an incitation to encode protocols, Dolev-Yao intruders, security assumptions, and even security properties, using first-order logic (Weidenbach, 1999; Blanchet, 2001). Cryptographic protocols can also be analyzed using tree automata (Comon et al., 1997), as shown in (Monniaux, 1999; Goubault-Larrecq, 2000), or using set constraints (Comon et al., 2001; Amadio and Charatonik, 2002), or using rewrite systems (Genet and Klay, 2000). A unified view, relating tree automata, set constraints and first-order logic, is provided in Goubault-Larrecq (2002c, 2004).

Extensions to encryption primitives obeying algebraic laws are now being explored by several researchers. The first decision procedures for classes of clause sets modulo equational intruders including the theory of associativity and commutativity of $\oplus$ are, to our knowledge, due to Goubault-Larrecq and Verma (2002). This includes the theory of exclusive or, and was illustrated on a model of the IKA.1 protocol (Steiner et al., 2000) for finitely many principals, using the e, $\oplus$, and $\mathbf{0}$ symbols described in the introduction; see also (Verma, 2003a).

Comon-Lundh and Cortier (2003) apply resolution techniques to the analysis of certain classes of clause sets in the presence of exclusive-or, that is, of a $\oplus$ operation that is associative, commutative, has a unit $\mathbf{0}$, and obeys the cancellation law $M \oplus M = 0$. Just like we do here, they restrict the use of resolution, and add new rules to retain completeness. They show how this can be used to model and analyze a simple protocol using exclusive-or. The complexity is prohibitive, though, and no implementation exists of their technique. The same can be said of the automata-theoretic procedures of Goubault-Larrecq and Verma (2002) and Verma (2003c,b), although the complexity of the latter are mostly unknown yet.

These procedures are *exact*, in the sense that given an input clause set (possibly approximating the given protocol), they always return true if the clause set is satisfiable, and no otherwise. Our procedures are *approximate*: they always return, and if the answer is true then the input clause set is satisfiable, but if the answer is false, then we cannot conclude. This is in the spirit of abstract interpretation (see Cousot and Cousot (1992) for a nice introduction).

Accordingly, the complexity of our procedure is lower (exponential time, still), and has been implemented by the second author. We won't report on experience with exclusive or here, just because we didn't conduct any experiment on this theory. It should be clear that our techniques apply to the case of exclusive or as well, and on other theories, too.

We shall report on results we obtained with the group key exchange protocol IKA.1 (Steiner et al., 2000). This protocol, and several others related to it, were scrutinized already by Pereira and Quisquater (2001). Their conclusions agree with ours. However, past experience has shown that verifying protocols by hand, even when proofs have been published in refereed conferences, is not enough. The most striking example is, of course, the (in)famous Needham-Schroeder public-key protocol (Needham and Schroeder, 1978), which was only found vulnerable, using formal methods, 17 years later (Lowe, 1996).

Since the work of Goubault-Larrecq and Verma (2002), several authors have proposed formal models of the Diffie-Hellman primitive. As indicated in the introduction, our model is abstract, and is based on a unary function e, a binary associative-commutative operation $\oplus$, with unit $\mathbf{0}$. The work of Chevalier et al. (2003) is based on a more precise model, as they use a binary function Exp instead of our unary e. $Exp(M, N)$ is meant to denote M to the exponent N, so our $e(N)$ can be seen as $Exp(g, N)$ for a fixed generator g. This is apparently richer, but no equation is included that relates applications of Exp to different generators. In particular, the equation $Exp(g, N) \oplus Exp(g', N) = Exp(g \oplus g', N)$ is not considered (remember that in the implementation of Diffie-Hellman using modular exponentiation, $\oplus$ is multiplication). If a fixed generator g is used throughout, then we believe that their model and ours are equivalent. The only difference would be that, whereas Chevalier et al. reason modulo the equation $Exp(g, M \oplus N) = Exp(Exp(g, M), N)$, we dispense with it and replace it by a deduction rule stating that, once you know $e(M)$ and N, then you know $e(M \oplus N)$. Replacing certain equations by deduction rules was explicitly used in Blanchet (2001), typically for equations of the form $\texttt{decrypt}(\texttt{encrypt}(M, K), K^{-1}) = M$. This trick is already the basis of all equational reasoning in NRL (Meadows, 1996). We shall call this the *Meadows trick*. As noticed e.g. in Kapur et al. (2003), while this works for some equational theories, this definitely fails with associativity and commutativity.

By far the most faithful model of modular exponentiation today is the $\mathcal{E}'_{3,5}$ model of Kapur et al. (2003), which includes all of Chevalier et al.'s equations, plus others, including the above formula $Exp(g, N) \oplus Exp(g', N) = Exp(g \oplus g', N)$. To be precise, Kapur et al. use two distinct symbols, $\cdot$ and $\circ$, for multiplication outside of exponents, and inside exponents respectively, which need not be related; $\cdot$ is an Abelian group law, while $\circ$ is a commutative monoid law. Several restrictions, all extending Chevalier et al.'s model and ours, are shown to give rise to a decidable unification problem, using extensions of Gröbner basis techniques. This entails that reachability properties (e.g., secrecy) are decidable in this model, since resolution then terminates provided only non-looping protocols with only finitely many sessions are considered. Chevalier et al. (2003) shows that the problem (in their slightly more restricted model) is even NP-complete in this case. We are not aware of any implementation of Chevalier et al.'s technique; while they claim that their procedure will find the known attack on IKA.1, no evidence is given that this was obtained by running any actual program.

One may think of models that would be even more faithful to modular exponentiation. We may consider the equation $Exp(g, N) \oplus Exp(g', N) = Exp(g \oplus g', N)$, i.e., make $\cdot$ and $\circ$ the same operation in Kapur et al.'s notation. We may also realize that there is another group law $+$ called addition modulo p, such that $Exp(g, M) \oplus Exp(g, N) = Exp(g, M + N)$. Or that $+$ and $\oplus$ together form a commutative ring, even a field when p is prime. (But then this is not an equational theory any longer.) Or even that the Jacobi symbol $\left(\frac{Exp(M,N)}{p}\right)$ equals $Exp\left(\left(\frac{M}{p}\right), N\right)$; all this can be used by an intruder, if only to get one bit of information on the plaintext M in the latter case (Goldwasser and Bellare, 1999). We must stop this inflation at some point. The theories in Kapur et al. (2003), at the frontier of decidability and undecidability, seem to be a natural limit. However, our techniques are not limited by this fact, since we use approximations anyway... and will never need to unify any term modulo the given theory. (We only use unification as a tool in proving completeness.)

2.2 Resolution.

We shall make extensive use of resolution theorem proving techniques (Robinson and Voronkov, 2001). Using resolution to decide subclasses of first-order logic formulas was pioneered by Joyner Jr. (1976), and earlier by Maslov, see (Fermüller et al., 2001). Standard refinements of resolution used in this area are *hyperresolution* and *ordered resolution*. We shall use ordered resolution *with selection* (Bachmair and Ganzinger, 2001). Previous work on cryptographic protocol verification using this rule include Comon-Lundh and Cortier (2003), and mostly Blanchet (2001), where it is used as a preprocessing step.

Let us give definitions. Let $\succ$ be a strict stable ordering on atomic formulas. By *stable* we mean that if $P(s) \succ Q(t)$, then $P(s\sigma) \succ Q(t\sigma)$ for any substitution σ, where $t\sigma$ denotes the application of σ to the term t. (Wlog, we restrict to unary predicate symbols.) A *literal* is either a *positive* literal $+P(t)$, or a *negative* literal $-P(t)$. A *clause* is a disjunction of literals $\pm_1 P_1(t_1) \vee \pm_2 P_2(t_2) \vee$

$\ldots \vee \pm_k P_k(t_k)$. A Horn clause is one containing at most one literal: we also write $P(t) \Leftarrow P_1(t_1), \ldots, P_n(t_n)$ for the *definite clause* $+P(t) \vee -P_1(t_1), \ldots, -P_n(t_n)$, and $\bot \Leftarrow P_1(t_1), \ldots, P_n(t_n)$ for the *goal clause* $-P_1(t_1), \ldots, -P_n(t_n)$. Let sel be a function mapping each clause to a subset of its negative literals.

Ordered resolution with selection is the rule that allows one to derive the conclusion (below the bar) provided we have already derived the premises (above), where:

$$\frac{\begin{array}{cc} C_1 \vee +A_{11} \vee \ldots \vee +A_{1m_1} & \\ \ldots & C' \vee -A'_1 \vee \ldots \vee -A'_n \\ C_n \vee +A_{n1} \vee \ldots \vee +A_{nm_n} & \end{array}}{(C_1 \vee \ldots \vee C_n \vee C')\sigma}$$

(i) $n \geq 1$, $m_1 \geq 1$, …, $m_n \geq 1$;
(ii) $\sigma = \text{mgu}\ (A_{11} \doteq \ldots \doteq A_{1m_1} \doteq A'_1, \ldots, A_{n1} \doteq \ldots \doteq A_{nm_n} \doteq A'_n)$, i.e., σ is the most general unifier (mgu) of the equations $A_{11} \doteq \ldots \doteq A_{1m_1} \doteq A'_1$, …, $A_{n1} \doteq \ldots \doteq A_{nm_n} \doteq A'_n$;
(iii) for every i, $1 \leq i \leq n$, $\text{sel}(C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}) = \emptyset$ and $A_{i1}, \ldots, A_{in_i}$ are maximal atomic formulae in $C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}$ with respect to $\succ$;
(iv) $\text{sel}(C' \vee -A'_1 \vee \ldots \vee -A'_n) = \{-A'_1, \ldots, -A'_n\} \neq \emptyset$, or $\text{sel}(C' \vee -A'_1 \vee \ldots \vee -A'_n) = \emptyset$ and A'_1, …, A'_n are maximal in $C' \vee -A'_1 \vee \ldots \vee -A'_n$ with respect to $\succ$.

For additional definitions, see (Bachmair and Ganzinger, 2001). It is implicit in the rule above that all premises have been renamed so that no two premises share any free variable. The right premise is called the *main premise*, all others are *side* premises. The conclusion is often called a *resolvent* of the premises.

In the case of Horn clauses, this simplifies to the rule:

$$\frac{A_1 \Leftarrow H_1 \quad \ldots \quad A_n \Leftarrow H_n \quad A \Leftarrow H, A'_1, \ldots, A'_n}{(A \Leftarrow H, H_1, \ldots, H_n)\sigma}$$

where H, H_1, …, H_n are *bodies*, i.e., sets of atomic formulas, comma denotes union of such sets, and the following conditions are met:

(i) $n \geq 1$;
(ii) $\sigma = \text{mgu}\ (A_1 \doteq A'_1, \ldots, A_n \doteq A'_n)$;
(iii) for every i, $1 \leq i \leq n$, $\text{sel}(A_i \Leftarrow H_i) = \emptyset$ and A_i is maximal in $A_i \Leftarrow H_i$ with respect to $\succ$;
(iv) letting C be $A \Leftarrow H, A'_1, \ldots, A'_n$, $\text{sel}(C) = \{-A'_1, \ldots, -A'_n\} \neq \emptyset$, or $\text{sel}(C) = \emptyset$ and A'_1, …, A'_n are maximal in C with respect to $\succ$.

This rule is *sound*, i.e., every conclusion is a consequence of the premises; in particular, if the empty clause $\perp$ is derivable from a given set of clauses S, then S is unsatisfiable. It is also *complete*: if S is unsatisfiable, then one can derive $\perp$ from S in finitely many steps of ordered resolution with selection.

It is folklore that soundness and completeness still hold when terms are taken modulo some equational theory $\mathcal{E}$, provided σ is taken to be any member of a *complete set of unifiers* (csu) csu $(A_1 \doteq A'_1, \ldots, A_n \doteq A'_n)$ in condition *(ii)*, and $\succ$ is *compatible with* E, meaning that if s_1, s_2 are equal mod E, if t_1, t_2 are equal mod E, and $s_1 \succ t_1$ then $s_2 \succ t_2$. This was already the case for other refinements of resolution (Plotkin, 1972). Such a csu always exists, but needs not be finite or even computable. One can compute a finite one for the theory of associativity and commutativity (AC), resp. with unit (ACU) (Stickel, 1981; Fagès, 1984).

Independently of equational reasoning, soundness and completeness are preserved when tautologies and various forms of subsumed clauses are removed, at any moment (preferably at the earliest) (Bachmair and Ganzinger, 2001). More importantly for making resolution terminate, we shall use rules of *splitting*. A clause of the form $C \vee C'$, where C and C' are non-empty clauses that share no free variable, is called *splittable*. Given a set of clauses $S \cup \{C \vee C'\}$, where $C \vee C'$ is splittable, the standard version of splitting (Weidenbach, 2001) then considers showing that both $S \cup \{C\}$ and $S \cup \{C'\}$ are unsatisfiable to conclude that $S \cup \{C \vee C'\}$ is. Each resulting clause set is called a *branch*. (Resolution with splitting is then indeed a form of tableaux.) For example, consider the following set of clauses:

$$\begin{array}{rr}
(a)\quad P(f(x,y)) \Leftarrow P_1(x), P_2(y) & (b)\quad Q_1(x) \Leftarrow P(f(x,y)), Q_2(y) \\
(c)\quad P_2(g(g(y))) \Leftarrow P_2(y) & (d)\quad P_2(g(a)) \\
(e)\quad Q_2(g(g(g(y)))) \Leftarrow Q_2(y) & (f)\quad Q_2(g(g(a))) \\
(g)\quad \perp \Leftarrow Q_1(g(a)) & (h)\quad P_1(g(x))
\end{array}$$

When the two clauses (a) and (b) are resolved on $P(f(x,y))$, we generate the clause $Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y)$, which is a splittable disjunction. The current branch then splits in two branches, on which reasoning proceeds independently:

$$(i)\quad Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y) \quad \text{by } (a), (b)$$

$$\begin{array}{c|c}
(j)\quad Q_1(x) \Leftarrow P_1(x) & (k)\quad \perp \Leftarrow P_2(y), Q_2(y) \\
(l)\quad \perp \Leftarrow P_1(g(a)) \quad \text{by } (j), (g) & (m)\quad \perp \Leftarrow P_2(y), Q_2(g(g(y))) \quad \text{by } (k), (c) \\
\perp \quad \text{by } (l), (h) & (n)\quad \perp \Leftarrow P_2(g(y)), Q_2(y) \quad \text{by } (m), (e) \\
 & (o)\quad \perp \Leftarrow P_2(y), Q_2(g(y)) \quad \text{by } (n), (c) \\
 & (p)\quad \perp \Leftarrow Q_2(g(g(a)) \quad \text{by } (o), (d) \\
 & (q)\quad \perp \quad \text{by } (p), (f)
\end{array}$$

Splitting may be applied at any time without breaking soundness or completeness. Because this makes our proofs slightly easier, we shall use Riazanov and Voronkov's special brand of splitting (Riazanov and Voronkov, 2001; Voronkov, 2001). We make it precise below, and call it *splittingless splitting* to distinguish it from ordinary splitting. The idea is that when $C \vee C'$ is splittable, then it is equivalent to $\exists q \cdot (q \Rightarrow C) \wedge (\neg q \Rightarrow C')$, where q is a fresh propositional symbol.

$$\frac{C \vee C'}{\begin{array}{c} C \vee -\ulcorner C' \urcorner \\ +\ulcorner C' \urcorner \vee C' \end{array}}$$

We make this formal as follows (Roger, 2003; Goubault-Larrecq, 2003). We first define formally what it means to create fresh propositional symbols. Fix a set $\mathcal{P}$ of predicate symbols. A *$\mathcal{P}$-clause* is any clause whose predicate symbols are all from $\mathcal{P}$. These will be our ordinary clauses. Let then $\mathcal{Q}$ be some set of zero-ary predicate symbols disjoint from $\mathcal{P}$, in one-to-one correspondence with the set of $\mathcal{P}$-clauses modulo renaming: for each $\mathcal{P}$-clause C, let $\ulcorner C \urcorner$ be a symbol in $\mathcal{Q}$, so that $\ulcorner C \urcorner = \ulcorner C' \urcorner$ iff there is a renaming ϱ such that $C = C'\varrho$. These will be our fresh symbols; however notice that we allow ourselves to reuse to same symbol $q = \ulcorner C' \urcorner$ when we meet the same clause C' twice. The rule of *splittingless splitting* is shown on the right, where C and C' are two non-empty subclauses sharing no variable, where C' is restricted to be a $\mathcal{P}$-clause, and C is required to contain at least an atom $P(t)$ with $P \in \mathcal{P}$.

The effect of the rule is to *replace* $C \vee C'$ by the two clauses $C \vee -\ulcorner C' \urcorner$ and $+\ulcorner C' \urcorner \vee C'$ in conclusion. Intuitively, $\ulcorner C' \urcorner$ is a propositional symbol that abbreviates the negation of C', i.e., that is false exactly when C' is valid. For example, the clause set (a)–(h) above generates the following clauses, when using resolution and splittingless splitting:

$$\begin{array}{ccl}
(i) & Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y) & \text{by } (a), (b) \\
(j') & Q_1(x) \Leftarrow P_1(x), q & \\
(k') & q \Leftarrow P_2(y), Q_2(y) & \\
(m') & q \Leftarrow P_2(y), Q_2(g(g(y))) & \text{by } (k'), (c) \\
(n') & q \Leftarrow P_2(g(y)), Q_2(y) & \text{by } (m'), (e) \\
(o') & q \Leftarrow P_2(y), Q_2(g(y)) & \text{by } (n'), (c) \\
(p') & q \Leftarrow Q_2(g(g(a))) & \text{by } (o'), (d) \\
(q') & q & \text{by } (p'), (f) \\
(j'') & Q_1(x) \Leftarrow P_1(x) & \text{by } (q'), (j') \\
(l') & \bot \Leftarrow P_1(g(a)) & \text{by } (j''), (g) \\
\bot & \text{by } (l'), (h) &
\end{array}$$

where $q = \ulcorner \bot \Leftarrow P_2(y), Q_2(y) \urcorner$. Note that steps (k')–(q') simulate exactly the right branch (k)–(q) of the tableau presented above, and the steps following (j'') simulate the left branch (as claimed by Voronkov (2001)).

Ordered resolution with selection, using $\mathcal{E}$-unification, is sound and complete, even when splittingless splitting is applied eagerly (i.e., when both rules can be applied, apply splittingless splitting), provided $\succ$ is a stable ordering such that $P(t) \succ q$ for every $P \in \mathcal{P}$, $q \in \mathcal{Q}$. (We say that $\succ$ is *admissible*.)

Proofs are omitted because these results can hardly be doubted, and for lack of space. See Goubault-Larrecq (2003) for a proof in the non-equational case (when $\mathcal{E}$ is the empty theory), and Roger (2003) for a proof in a more general case, including all equational theories.

In the sequel, we shall always use a special form of splittingless splitting, which we call *ϵ-splitting*: this is the special case where C' is a *negative block* $-P_1(x) \vee \ldots \vee -P_n(x)$ ($n \geq 1$; the variable x is the same in each literal), and where $C \vee C'$ is Horn. This is the brand of splitting we used in the example above. The ϵ-splitting rule can be reexplained as the one that replaces any clause $A \Leftarrow H, P_1(x), \ldots, P_n(x)$, where x is not free in A or H, by the two clauses $A \Leftarrow H, q$ and $q \Leftarrow P_1(x), \ldots, P_n(x)$, where $q = \ulcorner -P_1(x) \vee \ldots \vee -P_n(x) \urcorner$; in effect, this defines q as being true if and only if there is a term satisfying all of $P_1, \ldots, P_n$ in the least Herbrand model (if any exists).

3 Diffie-Hellman-like Protocols

3.1 The IKA.1 Protocol

Consider the initial key agreement protocol IKA.1 (Steiner et al., 2000) (formerly known as GDH.2), used to create an initial group key in the CLIQUES protocol suite. The goal is for a group of principals $\mathcal{M}_1, \ldots, \mathcal{M}_k$ to obtain a common key that they can use for further communication. This key should be unavailable to external eavesdroppers. (An *eavesdropper* is an intruder who may only listen to communication channels, but not forge messages, remove messages or redirect channels.) Moreover, no principal should be able to decide of the value of the key for the others; in general, no proper subset of the principals should be able to collude to create the common key.

We study the IKA.1 protocol in particular because it has attracted some attention recently. IKA.1 is based on a Diffie-Hellman scheme (Diffie and Hellman, 1976), which works as follows. We take the standpoint of a Dolev-Yao-like model (Dolev and Yao, 1983), where terms—here, modulo AC—are used to denote messages. As in the introduction, let e be a unary function symbol, and $\oplus$ a binary, associative and commutative symbol, and let $\mathbf{0}$ be a constant. Technically, we should impose that $\mathbf{0}$ is a unit for $\oplus$, but our techniques to come would not adapt right away; we shall get around this by using the Meadows trick, partially. Let also *cons* be a binary function, and *nil* be a constant, used to represent lists. We shall also use other constants, to be introduced later. We abbreviate $cons(M_1, cons(M_2, \ldots, cons(M_n, nil)\ldots))$ as $M_1; M_2; \ldots; M_n$. For simplicity, assume we have 3 members in the group, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$.

First, IKA.1 starts with a so-called *upflow* phase: $\mathcal{M}_1$ sends $\mathcal{M}_2$ the message $e(N_1)$, where N_1 is a fresh nonce; N_1 is modeled, as usual (Monniaux, 1999), as a new constant. Then $\mathcal{M}_2$ sends $e(N_2); e(N_1); e(N_1 \oplus N_2)$ to $\mathcal{M}_3$, where N_2 is another fresh nonce (modeled as another new constant $N_2 \neq N_1$). This is possible due to our assumptions that anybody can build $e(M)$ from M and $e(M \oplus M')$ from $e(M)$ and M'.

Once this is done, $\mathcal{M}_3$ starts the *downflow* phase, and broadcasts $e(N_2 \oplus N_3); e(N_1 \oplus N_3)$, from which all members can compute the group key $e(N_1 \oplus N_2 \oplus N_3)$. ($N_3$ is a third fresh nonce created by $\mathcal{M}_3$.)

We shall assume in the following discussion that each participant $\mathcal{M}_i$ $(i \geq 2)$ checks whether the list of messages it got from $\mathcal{M}_{i-1}$ in the upflow phase contains the trivial message $e(\mathbf{0})$, and if so, refuses to proceed. Otherwise, $\mathcal{M}_2$ could be caught receiving $e(\mathbf{0})$ just to send back $e(N_2); e(\mathbf{0}); e(\mathbf{0} \oplus N_2)$, that is, $e(N_2); e(\mathbf{0}); e(N_2)$ modulo ACU. If this were possible, then the protocol would fail, since N_1 would be lost in the process, hence the group key would be computed independently of N_1. This is a perfectly reasonable assumption in practice: any serious implementation of IKA.1 will include similar checks. This will also allow us to use clauses modulo AC instead of modulo ACU.

3.2 Modeling IKA.1 with a Fixed Number of Sessions

All possible interleaved executions of the protocol can be described using Horn clauses modulo AC. Let us write selected clauses from this set.

To model communication, introduce predicates ch_C for each configuration C that is reachable in an interleaved run of $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$. Here a configuration is a triple of numbers, each one stating which step of the protocol the corresponding principal was stationed at. The formula $ch_C(M)$ is meant to hold if and only if M is a possible message present on some communication channel when in configuration C. We also create distinct predicates I_C such that $I_C(M)$ is meant to hold when M is deducible, in a sense inspired by Dolev and Yao (1983), from all messages present on the communication channels at or before configuration C.

As such, for every reachable configuration C, we generate the clauses:

$$I_C(\mathbf{0}) \qquad \text{Intruder knows } \mathbf{0} \tag{1}$$
$$I_C(e(x)) \Leftarrow I_C(x) \qquad \text{Intruder can} \tag{2}$$
$$I_C(e(x \oplus y)) \Leftarrow I_C(e(x)), I_C(y) \qquad \text{exponentiate} \tag{3}$$
$$I_C(nil) \qquad \text{Intruder knows the empty list} \tag{4}$$
$$I_C(cons(x, y)) \Leftarrow I_C(x), I_C(y) \quad \text{Intruder can build lists} \tag{5}$$
$$I_C(x) \Leftarrow I_C(cons(x, y)) \qquad \text{Intruder can read heads} \tag{6}$$
$$I_C(y) \Leftarrow I_C(cons(x, y)) \qquad \text{Intruder can read tails} \tag{7}$$

We shall define several intrusion models. This will have an impact on the clauses we shall generate to define ch_C. Whatever the model, we shall generate the clauses

$$I_C(x) \Leftarrow ch_C(x) \text{ Intruder spies on every channel} \tag{8}$$
$$I_C(x) \Leftarrow I_{C'}(x) \text{ Intruder remembers past messages} \tag{9}$$

for any reachable configuration C, and where C' is the predecessor of C (if any) in (9).

The most benign model from a security viewpoint is the *pure eavesdropper model*, where the intruder does not interfere with communication, except for remembering all messages exchanged by the honest principals $\mathcal{M}_i$. If $\mathcal{M}_i$ sends M under some conditions H, then we generate the clause $ch_C(M) \Leftarrow H$. If $\mathcal{M}_i$ waits for some message M to be read before doing some action described by a clause $A \Leftarrow H$, then we generate $A \Leftarrow H, ch_C(M)$. (This will be described in more detail below.) In the pure eavesdropper model, this will be all.

In the *copycat model*, the intruder is also able to replay old messages, and to divert communication channels. We generate the additional clause

$$ch_C(x) \Leftarrow ch_{C'}(x) \tag{10}$$

for any reachable configuration C with predecessor C'. This specifies that every message put on some channel remains on this channel, and can be read at any time in the future, and replayed as many times as we wish.

In the *Dolev-Yao model*, so named because it is closest to that of Dolev and Yao (1983), instead of (10), we generate the clause

$$ch_C(x) \Leftarrow I_C(x) \text{ Intruder has complete control over channels} \tag{11}$$

In other words, anything read from any channel is directly forged by the intruder from past messages. As far as security is concerned, this is the model giving the most abilities to the intruder. Any attack in all previous models can be played in this model.

The final required ingredient is the specification of the contents of the communication channels in the initial configuration C_0: we assume that some predicate ch_{C_0} has been defined by some set of Horn clauses. We further restrict the pure eavesdropper model to be such that the channel is empty in configuration C_0: just don't produce any clause defining ch_{C_0}.

Starting from C_0, execution may proceed by letting $\mathcal{M}_1$ send its upflow message to $\mathcal{M}_2$, letting the whole system progress to some new configuration C_1 (and C_0 is its predecessor):

$$ch_{C_1}(e(N_1)) \text{ Intruder gets } \mathcal{M}_1\text{'s message} \tag{12}$$

Note that ch_{C_1} holds of exactly one message in the pure eavesdropper model. This will be an invariant of our description: for any reachable configuration C, there will be at most one ground term M such that $ch_C(M)$ holds—the *contents* of the channel. In the copycat and Dolev-Yao models, $ch_C(M)$ may be true for several messages M, because of clauses (10), resp. (11).

Let us write what happens if $\mathcal{M}_2$ runs next, sending its own message to $\mathcal{M}_3$:

$$ch_{C_2}(e(N_2); e(x); e(x \oplus N_2)) \Leftarrow ch_{C_1}(e(x)) \tag{13}$$

Clause (13) means that $\mathcal{M}_2$ reads the message from $\mathcal{M}_1$ first; this should be $e(N_1)$, but $\mathcal{M}_2$ can only check that it is $e(x)$ for some x; also, the only way it can get $e(x)$ is by querying the channel through ch_{C_1}. Then $\mathcal{M}_2$ should build $e(N_2); e(N_1); e(N_1 \oplus N_2)$. Since the variable x should contain N_1, actual implementations build $e(N_2); e(x); e(x \oplus N_2)$, and send it to the intruder in the new configuration C_2. (Note that this clause does not represent the case where the x received is $\mathbf{0}$; in this case, we should either switch to the theory ACU, or generate the additional clause $ch_{C_2}(e(N_2); e(\mathbf{0}); e(N_2)) \Leftarrow ch_{C_1}(e(x))$. As we said earlier, we assume that the implementation checks that x is not $\mathbf{0}$, which dispenses us from doing either.)

The downflow message from $\mathcal{M}_3$ gives rise to the clause:

$$ch_{C_3}(e(x \oplus N_3); e(y \oplus N_3)) \Leftarrow ch_{C_2}(e(x); e(y); e(z)) \tag{14}$$

Now the secrecy requirement on, say, $\mathcal{M}_1$'s view of the group key is that

$$\perp \Leftarrow ch_{C_3}(cons(e(x), z')), I_{C_3}(e(x \oplus N_1)) \tag{15}$$

Indeed, $\mathcal{M}_1$'s view of the group key is $e(x \oplus N_1)$, where the message broadcasted by $\mathcal{M}_3$ is $e(x); y$, i.e., where $ch_{C_3}(e(x); y)$ holds (reminder: if this message is not forged, then $x = N_2 \oplus N_3$). Clause (15) states that this view $e(x \oplus N_1)$ is not known to the intruder in configuration C_3 (and therefore neither in C_1 or C_2.)

There are many other possible interleavings, whose description we leave to the reader. The important point is that, if there is an attack, then the resulting set of clauses S_0 will be unsatisfiable: an attack will provide a derivation of $\perp$. Hence, if S_0 is satisfiable, then the protocol is secure. This observation is due to Selinger (2001) in the non-equational case.

3.3 Transformations of Clauses

It will be useful in the sequel to work on restricted forms of clauses. The main reason is that the satisfiability of general clause formats is undecidable, but some transformations can be used to convert clauses to a decidable format, possibly losing some information. The prototype of this is given by Frühwirth et al. (1991). We describe a very similar transformation, inspired from Goubault-Larrecq (2002a).

The main transformation we use is *flattening*. The purpose of flattening is to transform the input clause set S_0 into a set of so-called *flat* clauses S_1, in which all predicate symbols are applied either to variables or to *flat terms* $f(x_1, \ldots, x_n)$, where f is a function symbol and $x_1, \ldots, x_n$ are variables.

Let S be a set of clauses. If there is a clause $C = C_0 \vee \pm P(t)$ in S where t is neither a variable nor a flat term, i.e., $t = f(t_1, \ldots, t_n)$, then:

(1) If $\pm$ is $-$, create n fresh predicate symbols P_j, $1 \leq j \leq n$, and replace C in S by the $n + 1$ clauses:

$$C_0 \vee -P_1(t_1) \vee \ldots \vee -P_n(t_n) \tag{16}$$
$$-P(f(X_1, \ldots, X_n)) \vee +P_j(X_j) \qquad (1 \leq j \leq n) \tag{17}$$

where $X_1, \ldots, X_n$ are fresh variables, not necessarily distinct, but such that $t_i \neq t_j$ implies $X_i \neq X_j$, $1 \leq i \leq j \leq n$;

(2) If $\pm$ is $+$, create n fresh predicate symbols P_j, $1 \leq j \leq n$, and replace C in S by the $n + 1$ clauses:

$$+P(f(X_1, \ldots, X_n)) \vee -P_1(X_1) \vee \ldots \vee -P_n(X_n) \tag{18}$$
$$C_0 \vee +P_j(t_j) \qquad (1 \leq j \leq n) \tag{19}$$

Write $S \rightsquigarrow S'$ if S' is obtained by such a transformation. It is easy to see that $\rightsquigarrow$ terminates: let $|t|$ be the size of term t, i.e., $|x| = 0$ for every variable x,

$|f(t_1,\ldots,t_n)| = 1 + |t_1| + \ldots + |t_n|$, and the size of the clause $\pm_1 P_1(t_1) \vee \ldots \vee \pm_n P_n(t_n)$ be $\sum_{i=1}^n |t_i|$, then these rules replace at least one clause by clauses of lower sizes.

Furthermore, every normal form is a set of flat clauses. Moreover, if $S \rightsquigarrow S'$, then S' logically implies S: just resolve (16) and (17), or (18) and (19) on the new atoms and you get back C. So, if S' is satisfiable, then so will be S. In particular, letting S_1 be any $\rightsquigarrow$-normal form of S_0, if S_1 is satisfiable, then S_0 will be too, and the protocol will be secure. In general, the converse is wrong, so this is an *abstraction*. It turns out that, in the case of the clauses of Section 3.2, this abstraction is exact in all three models (pure eavesdropper, copycat, Dolev-Yao): see Section 3.4.

A slight optimization of flattening in the presence of an AC symbol $\oplus$ is to modify the definition of flat terms so that any sum of variables $x_1 \oplus x_2 \oplus \ldots \oplus x_n$ is considered a flat term.

We may extend flattening so as to apply the transformation rules (1) and (2) above even when t is of the form $f(x_1,\ldots,x_n)$ where x_1, …, x_n are variables, provided some additional conditions are met. (Otherwise, the procedure would loop on clauses of the form (17) and (19).)

One typical case is when the rest C_0 of the clause contains another literal $\pm P'(g(y_1,\ldots,y_m))$ where the set $\{y_1,\ldots,y_m\}$ of variables is different from $\{x_1,\ldots,x_n\}$ but $\{y_1,\ldots,y_m\} \cap \{x_1,\ldots,x_n\} \neq \emptyset$, or when $f \neq g$. Normalizing clauses with this relaxed definition of flattening ensures that the resulting clauses are either *blocks*, i.e., clauses of the form $\pm_1 P_1(x) \vee \ldots \vee \pm_n P_n(x)$ (for the *same* variable x; we abbreviate such blocks $B(x)$); or *complex clauses* (in the terminology of Goubault-Larrecq (2002a)), i.e., clauses of the form $\bigvee_{i=1}^m \pm_i P_i(f(x_1,\ldots,x_n)) \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$, where B_1, …, B_n are blocks and $m \geq 1$ (with the *same* f and the same set of variables $x_1,\ldots,x_n$ as arguments of f); or disjunctions of such clauses, not sharing any free variable. When $B(x)$ is a negative block $-P_1(x) \vee \ldots \vee -P_n(x)$, we shall write $-B(x)$ for the set $P_1(x),\ldots,P_n(x)$ on the right of $\Leftarrow$ in Horn clauses.

Blocks and complex clauses correspond exactly to a format of positive set constraints, with equality constraints between brothers, when reasoning modulo the empty theory; Goubault-Larrecq (2002a) uses this observation to generalize positive set constraints to higher-order terms. That set constraints correspond to a decidable class of first-order formulas, namely the monadic class, was observed by Bachmair et al. (1993).

Sets of flattened Horn clauses modulo $\mathcal{E}$ are, up to small variations, the alternating two-way tree automata modulo $\mathcal{E}$ of Verma (2003c,b). We shall call them *automata clauses* for short. In other words, automata clauses are comprised of blocks and complex clauses.

The satisfiability of alternating two-way tree automata modulo AC is undecidable, but dropping alternation allows one to regain decidability, with non-primitive complexity in the AC case (Verma, 2003a). We use flattened Horn clauses in this paper simply because these are the cases on which the techniques of Section 4 work.

3.4 Flattening the IKA.1 Clauses

All clauses of Section 3.2 but a few are automata clauses. E.g., clauses (1), (2), (4), (5), (6), (7), (8), (9), (10), and (11) are automata clauses.

The remaining ones can be rewritten in the form of automata clauses by introducing auxiliary predicate symbols, that is, by flattening. Clause (3) is flattened to:

$$Q_C(x) \Leftarrow I_C(e(x)) \qquad Q'_C(x \oplus y) \Leftarrow Q_C(x), I_C(y) \qquad I_C(e(z)) \Leftarrow Q'_C(z)$$

where Q_C and Q'_C are fresh. Note that we do not lose any information here: the clauses above state exactly the same thing as $I_C(e(x \oplus y)) \Leftarrow I_C(e(x)), I_C(y)$, provided we define Q_C so that Q_C holds of every term t such that $I_C(e(t))$ holds, and Q'_C holds of every sum of terms t and t' such that t satisfies Q_C and t' satisfies I_C.

A similar observation holds of clauses (12) and (15). We can flatten the latter as

$$R(z) \Leftarrow ch_{C_3}(cons(z, z')) \qquad R'(x) \Leftarrow R(e(x)) \qquad R''(x \oplus N_1) \Leftarrow R'(x)$$
$$R'''(e(z)) \Leftarrow R''(z) \qquad \perp \Leftarrow R'''(z), I_{C_3}(z)$$

where R, R', R'', R''' are fresh. There are other flattenings. We choose this one because it avoids generating so-called intersection (a.k.a., alternation) clauses, of the form $P(x) \Leftarrow Q(x), Q'(x)$. This would be important if we were to use Verma (2003a)'s techniques, as this shows that we do not need alternation, a cause of undecidability, for this clause.

The only clauses where flattening is approximate are (13) and (14). The clauses

$$valx(x) \Leftarrow ch_{C_1}(e(x)) \qquad ch_{C_2}(x_1; x_2; x_3) \Leftarrow P_1(x_1), P_2(x_2), P_3(x_3)$$
$$P_1(e(N_2)) \qquad P_2(e(x)) \Leftarrow valx(x) \qquad P_3(e(x \oplus N_2)) \Leftarrow valx(x)$$

are equivalent to (13) in the Dolev-Yao model, in the copycat model (where sending lists of elements or sending each element separately has the same net effect) and in the pure eavesdropper model (where any ch_C recognizes at most one value anyway). These can be converted to automata clauses:

$$valx(x) \Leftarrow ch_{C_1}(e(x))$$
$$ch_{C_2}(cons(x_1, y)) \Leftarrow P_1(x_1), ch^1_{C_2}(y) \qquad ch^1_{C_2}(cons(x_2, y)) \Leftarrow P_2(x_2), ch^2_{C_2}(y)$$
$$ch^2_{C_2}(cons(x_3, y)) \Leftarrow P_3(x_3), ch^3_{C_2}(y) \qquad ch^3_{C_2}(nil)$$
$$P_1(e(x)) \Leftarrow P^1_1(x) \qquad P^1_1(N_2)$$
$$P_2(e(x)) \Leftarrow valx(x)$$
$$P_3(e(x)) \Leftarrow P^1_3(x) \qquad P^1_3(x \oplus N_2) \Leftarrow valx(x)$$

where all newly introduced predicates are fresh. By a similar argument clause (14) can be converted to automata clauses:

$$P'_1(x_1) \Leftarrow ch_{C_2}(cons(x_1, z_1)) \qquad P'_2(z_1) \Leftarrow ch_{C_2}(cons(x_1, z_1)) \qquad P'_5(nil)$$
$$P'_3(y_1) \Leftarrow P'_2(cons(y_1, z_1)) \quad P'_4(e(x \oplus N_3)) \Leftarrow P'_1(e(x)) \quad P'_6(e(y \oplus N_3)) \Leftarrow P'_3(e(y))$$
$$P'_7(cons(z, z')) \Leftarrow P'_6(z), P'_5(z') \qquad ch_{C_3}(cons(x', y')) \Leftarrow P'_4(x'), P'_7(y')$$

Apart from giving an illustration of flattening, this section is meant to show that the IKA.1 protocol with finitely many principals can be encoded exactly in the decidable fragment of sets of automata clauses modulo AC without alternation. While we won't make any use of this observation here, since we rely on approximate techniques anyway, it may be useful in further exact treatments of IKA.1.

4 Deciding Sets of Horn Clauses Modulo AC with Resolution

4.1 Preliminaries: Deciding the Non-Equational Case by Resolution

To decide sets of automata clauses, we use ordered resolution with selection, eager ϵ-splitting, and elimination of tautologies and forward subsumed clauses. To see why this is interesting, let us explain why this strategy decides the satisfiability of any finite set of automata clauses *modulo the empty theory*; we shall deal with the AC case later. Note that this satisfiability problem is exactly the satisfiability problem of definite positive set constraints with equality tests between brothers, see Goubault-Larrecq (2002a) for details. (The more general case of non-definite positive set constraints, i.e., the case of possibly non-Horn flattened clauses can also be handled this way.)

Choose the ordering $\succ$ so that $P(t) \succ q$ for every $P \in \mathcal{P}$ and $q \in \mathcal{Q}$, and $P(t) \succ P'(t')$ whenever t is a strict superterm of t', whatever $P, P' \in \mathcal{P}$ are. This is a stable ordering, hence an admissible ordering, so ordered resolution with selection is sound and complete if we use this ordering.

To make resolution terminate, define the selection function sel_0 as follows:

- if C contains a negative literal $-q$ with $q \in \mathcal{Q}$, then $\mathrm{sel}_0(C) = \{-q\}$;
- otherwise, let $\max(C)$ be the set of maximal literals in C for $\succ$, then define $\mathrm{sel}_0(C)$ as the subset of those negative literals in $\max(C)$.

For example, in clause (a) in the example of Section 2.2, namely $P(f(x,y)) \Leftarrow P_1(x), P_2(y)$, there is no negative literal $-q$, so apply the second item. The only maximal literal is $+P(f(x,y))$, and it is positive. So no literal is selected in this clause. By condition *(iii)* of the definition, this clause can be used as main premise; by condition *(iv)*, it cannot be used as side premise. In clause (b) $Q_1(x) \Leftarrow P(f(x,y)), Q_2(y)$, the only selected literal is $-P(f(x,y))$. Clause (b) can therefore be used as side premise but not as main premise. So ordered resolution with selection can be used in only one way on clauses (a) and (b), yielding clause (i) $Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y)$. This clause can be ϵ-split immediately, yielding clauses (j') $Q_1(x) \Leftarrow P_1(x), q$ and (k') $q \Leftarrow P_2(y), Q_2(y)$. Notice that, up to the addition of q literals, the resulting clauses are still automata clauses.

More formally, let $\mathcal{Q}_0$ be the set of all q literals of the form $\ulcorner B(x) \urcorner$, where $B(x)$ is any non-empty negative block. Since $B(x)$ is a block, it is a $\mathcal{P}$-clause. If $\mathcal{P}$ contains p predicate symbols, then $\mathcal{Q}_0$ contains $2^p - 1$ elements.

We claim that, given any finite set S of automatic $\mathcal{P}$-clauses (modulo the empty theory), every clause obtained from S by the above resolution strategy is Horn and of one of the following forms:

(1) a clause $C[\vee + q]$, where C is a block and $q \in \mathcal{Q}_0$ (the notation $C[\vee + q]$ meaning a clause of the form C or $C \vee +q$);
(2) a clause $C[\vee + q]$, where C is a complex $\mathcal{P}$-clause, and $q \in \mathcal{Q}_0$;
(3) or a clause $C \vee -q_1 \vee \ldots \vee -q_m[\vee + q]$, where C is a block, $q_1, \ldots, q_m, q \in \mathcal{Q}_0$ and $1 \leq m \leq a$, and a is the maximal arity of all function symbols;

Indeed:

- Resolvents of clauses of type **(1)** are again of type **(1)**.
- Any ordered resolution step with selection between clauses of type **(2)** must have as premises clauses of the form $C \vee +P(f(x_1, \ldots, x_m))$ and $C' \vee -P(f(x_1, \ldots, x_m))[\vee + q]$ (where $\pm P(f(x_1, \ldots, x_m))$ is maximal in both), yielding $C \vee C'[\vee + q]$; either the latter is of type **(2)** again if there is another literal $\pm Q(f'(x_1, \ldots, x_m))$ in C or in C', or $C \vee C'$ is a disjunc-

tion $B_1(x_1) \vee \ldots \vee B_m(x_m)$ of m blocks, as in the example of clauses (a) and (b) above; in the latter case, note that since $C \vee C'$ is Horn, at most one block $B_i(x_i)$ is non-negative: repeated ϵ-splitting then produces the clauses $+\ulcorner B_j(x_j)\urcorner \vee B_j(x_j)$ (which are of type **(1)**), for all $j \neq i$ such that $B_j(x_j) \neq \bot$, and the clause $B_i(x_i) \vee \bigvee_{j \neq i, B_j(x_j) \neq \bot} -\ulcorner B_j(x_j)\urcorner[\vee + q]$ (which is of type **(3)**).

- Any ordered resolution step with selection between a clause of type **(1)** and a clause of type **(2)** must be between clauses of the form $C \vee -P(x)[\vee + q]$ (type **(1)**) and $C' \vee +P(f(x_1, \ldots, x_m))$ (type **(2)**), or between clauses $C \vee +P(x)$ (type **(1)**) and $C' \vee -P(f(x_1, \ldots, x_m))[\vee + q]$ (type **(2)**), yielding $C[x := f(x_1, \ldots, x_m)] \vee C'[\vee + q]$ in both cases. This is of type **(2)** if C is non-empty or there is a literal $\pm Q(f'(x_1, \ldots, x_m))$ in C'; otherwise, this is a disjunction of m blocks (possibly with $+q$), yielding clauses of type **(1)** and **(3)** by ϵ-splitting, as in the previous case.
- Next, the only way we can resolve a type **(3)** clause with some other clause is to resolve on one of the selected literals $-q_i$, $1 \leq i \leq m$; this can only be done with a clause C containing $+q_i$, but this is impossible if C is of type **(1)** or **(2)** since q_i is not maximal in C, unless C is exactly the unit clause $+q_i$, in which case the resolvent is of type **(3)** or **(1)**; and this is impossible when C is of type **(3)**, since then $\mathrm{sel}_0(C) \neq \emptyset$.

As far as complexity is concerned, a quick computation shows that we generate at most an exponential number of clauses in the number p of predicates and the maximum arity a of functions, and that they have at most exponential size. So this resolution strategy terminates in exponential time. This is easily seen to be optimal: we have retrieved the result that the satisfiability of positive set constraints with equality tests between brothers is DEXPTIME-complete.

4.2 Why the AC Case is Harder

Unfortunately, resolving two clauses that use some AC symbol $\oplus$ generates bigger clauses in general, in such a way that we cannot bound their size.

In particular, imitating the argument of Section 4.1 by trying, say, to restrict to blocks and complex clauses (letting $f_1, \ldots, f_n$ be any function symbols, including AC symbols), fails. Consider for example the following clauses

$$P(x') \Leftarrow Q'_C(x' \oplus y') \qquad Q'_C(x \oplus y) \Leftarrow Q_C(x), I_C(y)$$

(We have already obtained the second clause by expanding clause (3) in Section 3.4.) One complete set of AC-unifiers of $x' \oplus y'$ and $x \oplus y$ consists of the following 7 substitutions, where x_1, x_2, y_1, y_2 are fresh, pairwise distinct variables:

$$(1)\ [x := x_1 \oplus y_1, y := x_2 \oplus y_2, x' := x_1 \oplus x_2, y' := y_1 \oplus y_2]$$

$$(2)\ [y := x' \oplus y_2, y' := x \oplus y_2] \qquad (3)\ [y := y' \oplus x_2, x' := x \oplus x_2]$$

$$(4)\ [x := x' \oplus y_1, y' := y \oplus y_1] \qquad (5)\ [x := y' \oplus x_1, x' := y \oplus x_1]$$

$$(6)\ [x' := y, y' := x] \qquad (7)\ [x' := x, y' := y]$$

We then get the following 7 resolvents:

$$(1)\ P(x_1 \oplus x_2) \Leftarrow Q_C(x_1 \oplus y_1), I_C(x_2 \oplus y_2)$$

$$(2)\ P(x') \Leftarrow Q_C(x), I_C(x' \oplus y_2) \qquad (3)\ P(x \oplus x_2) \Leftarrow Q_C(x), I_C(y' \oplus x_2)$$

$$(4)\ P(x') \Leftarrow Q_C(x' \oplus y_1), I_C(y) \qquad (5)\ P(y \oplus x_1) \Leftarrow Q_C(y' \oplus x_1), I_C(y)$$

$$(6)\ P(y) \Leftarrow Q_C(x), I_C(y) \qquad (7)\ P(x) \Leftarrow Q_C(x), I_C(y)$$

Clauses (2), (4), (6), and (7) split further, yielding

$$\begin{array}{ccc} P(x') \Leftarrow I_C(x' \oplus y_2), q_1 & q_1 \Leftarrow Q_C(x) & P(y) \Leftarrow I_C(y), q_1 \\ P(x') \Leftarrow Q_C(x' \oplus y_1), q_2 & q_2 \Leftarrow I_C(y) & P(x) \Leftarrow Q_C(x), q_2 \end{array}$$

respectively using ϵ-splitting, where $q_1 = \ulcorner -Q_C(x) \urcorner$, $q_2 = \ulcorner -I_C(y) \urcorner$. However, clauses (1), (3), and (5) are neither blocks nor complex clauses, because they contain literals whose sets of free variables do not contain all the free variables of the clause; this was an essential condition for decidability in the non-AC case of Section 4.1. It is probably futile to try to allow for more general complex clauses, until we reach a notion that is general enough; resolving these clauses with other clauses only produces larger and larger resolvents, involving larger and larger sums of variables.

4.3 Grey Zones, Grey Oracles

Instead, our action plan is as follows. Imagine for the moment that we are using just resolution to decide the satisfiability of automata clauses, modulo AC. Then any proof is a tree whose nodes are labeled by clauses; if resolution is applied to the side premises $C_1, \ldots, C_n$ and the main premise C, with conclusion C', then C' will label a node whose sons are labeled with $C_1, \ldots, C_n$, C. Splitting would complicate matters quite a lot here. If we overlook the problem with splitting for the moment, and if we ignore the necessity of using q literals, the arguments of Section 4.1 show that, as long as we deal with blocks and complex clauses with only *non-equational* function symbols (all but

$\oplus$ in our case), only finitely many clauses, either blocks or similar complex clauses, can be produced. As soon as $\oplus$ comes into play, we may get larger and larger clauses using only $\oplus$ as function symbol (so-called $\oplus$-*clauses*), see Section 4.2. But, if such clauses eventually participate in deriving the empty clause, it must be the case that one $\oplus$-clause thus derived eventually resolves with other clauses to get a conclusion C that is either directly the empty clause, or can resolve with complex clauses not containing $\oplus$. Since no term headed by $\oplus$ unifies with a term headed by f, with $f \neq \oplus$, and provided we only unify on maximal atoms, C can only be a disjunction of literals of the form $\pm P(x)$, with x a variable. Hence C must split into blocks.

This is tentatively pictured in Figure 1; the leaves of the derivation (at the top) are clauses in the initial clause set S. Resolution steps inside the white zones are those that we met in Section 4.1, and which must terminate since they only generate finitely many clauses. Resolution steps inside the grey zones produce arbitrarily many, arbitrarily large $\oplus$-clauses. This leads us to the following idea: instead of applying resolution inside the grey zones, try to *guess* the fat dots, which are the interface points between grey zones and white zones. Forbid resolution to act on $\oplus$-clauses (this prevents us from using resolution to derive clauses inside the grey zones), and compensate this by adding a rule that infers the fat dot clauses, at the bottom of grey zones, directly from the clauses at the top of grey zones: this is the *oracle rule*. At this point, we must admit that this does not seem practical at all; the perspective will gradually improve. Let us only notice however that the fat dots, which are splittable disjunctions of blocks, are only finitely many.

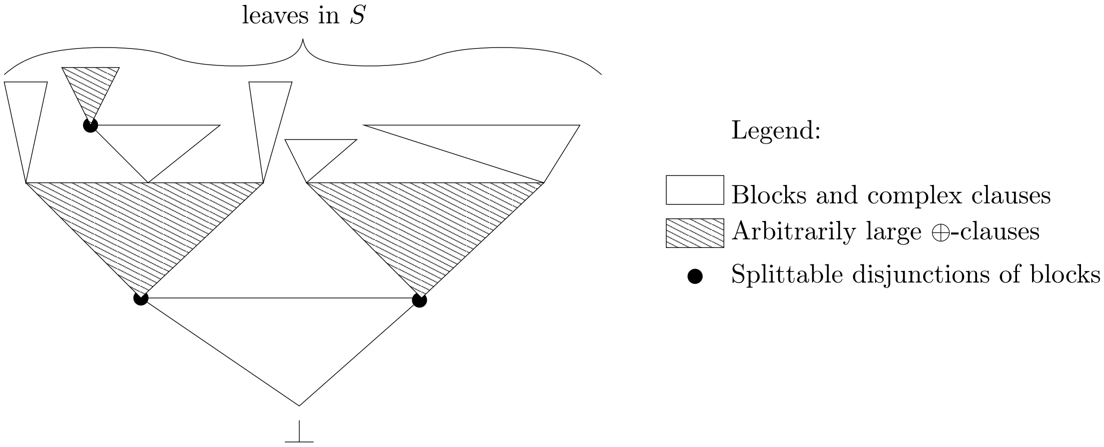

Fig. 1. Grey zones, fat dots, white zones

Let us just give the idea of a possible oracle. Inside a grey zone, we only have $\oplus$-clauses. In particular, no symbol other than $\oplus$ occurs. So all deductions made inside a grey zone are valid in any model of the sole associative-commutative symbol $\oplus$. E.g., consider $\mathbb{Z}/n\mathbb{Z}$, where $\oplus$ denotes addition modulo n. We then compute all instances of clauses given on entry to the grey zone inside the model; for example, the instances of $P(x \oplus y) \Leftarrow Q(x), R(y)$ in $\mathbb{Z}/2\mathbb{Z}$ are

$P(0) \Leftarrow Q(0), R(0)$; $P(1) \Leftarrow Q(0), R(1)$; $P(1) \Leftarrow Q(1), R(0)$; and $P(0) \Leftarrow Q(1), R(1)$. We then let the oracle simulate resolution at the level of first-order clauses by computing all resolvents on the instances over $\mathcal{M}$. From time to time, we examine a given inferred instance, and guess which fat dot clause it might be an instance of.

In the following sections, we make the intuition behind Figure 1 formal. Indeed, while we hope that the discussion above conveys the essential ideas, the arguments in this discussion are slightly wrong, and certainly cannot serve as proof. This will require several careful definitions, and gradual modifications to the basic ordered resolution with selection rule before we can state the fundamental oracle lemma. (Please fasten your seat belts: this is definitely technical. We have done our best to make it accessible.)

4.4 Scrutinizing the Resolution Rule

It is as easy (or as difficult) to deal with the special case of one $\oplus$ symbol as to deal with many equational theories at once, so let us introduce the corresponding generalization. Recall that a *signature* Σ is a set of function symbols, together with their arities. A Σ-*term*, or equivalently a term *on* Σ is any term whose function symbols are all taken from Σ. We say that a $\mathcal{P}$-clause is a Σ-*clause* if and only if it is of the form $\pm_1 P_1(t_1) \vee \ldots \vee \pm_k P_k(t_k)$ where $t_1, \ldots, t_k$ are Σ-terms. It is a *non-trivial* Σ-*clause* if and only if, additionally, not all of $t_1, \ldots, t_k$ are variables. Similarly, an equation $s \approx t$ is *on* Σ if and only if s and t are both on Σ. (We use the symbol $\approx$ for equations to distinguish it from true equality; we assume that $s \approx t$ and $t \approx s$ are the same equation.) An *equational theory on* Σ is a set E of equations on Σ. The notion of an equality $s \approx t$ provable from E, written $E \models s \approx t$ is defined as usual, inductively, by the rules $E \models s \approx s$ (reflexivity), $E \models s \approx t$ implies $E \models t \approx s$ (symmetry), $E \models s \approx t$ and $E \models t \approx u$ imply $E \models s \approx u$ (transitivity), $E \models s\sigma \approx t\sigma$ if $(s \approx t) \in E$ for any substitution σ (even mapping variables to terms that are *not* on Σ, but on a larger signature), and $E \models s_i \approx t_i$, $1 \leq i \leq n$, together imply $E \models f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n)$ (congruence). For example, let AC be the theory $\{x \oplus y \approx y \oplus x, (x \oplus y) \oplus z \approx x \oplus (y \oplus z)\}$ of associativity and commutativity of $\oplus$, a theory on the signature $\{\oplus\}$, then $\mathrm{AC} \models f(a) \oplus ((b \oplus c) \oplus f(c)) \approx (b \oplus c) \oplus (f(a) \oplus f(c))$.

We say that an equational theory E on Σ is *simple* if and only if there is a computable strict, stable ordering $\succ_{E,\Sigma}$ compatible with E such that $f(x_1, \ldots, x_n) \succ_{E,\Sigma} x_i$ for every i, $1 \leq i \leq n$, and every function symbol f (possibly outside Σ), where $x_1, \ldots, x_n$ are distinct variables. In other words, $\succ_{E,\Sigma}$ is a computable, compatible *simplification ordering*. AC and the empty theory are simple: take $\succ_{E,\Sigma}$ to be defined by $s \succ_{E,\Sigma} t$ if and only if $|s| > |t|$, where $|u|$ denotes the *size* of term u. $\mathrm{ACU} = \mathrm{AC} \cup \{x \oplus \mathbf{0} \approx x\}$ is not simple.

Note that, assuming there is at least one non-constant function symbol f, any simple equational theory E must be *collapse-free*, i.e., if $E \models s \approx t$, then either s and t are the same variable x, or neither s nor t is a variable. Indeed, assume otherwise: $E \models s \approx x$, where x is a variable and s is not. If x is free in s, then $s \succ_{E,\Sigma} x$ by simplicity, hence $s \succ_{E,X} s$ by compatibility, which is impossible. If x is not free in s, then $E \models s[x := f(x, \ldots, x)] \approx x[x := f(x, \ldots, x)]$, so since x is not free in s, $E \models s \approx f(x, \ldots, x)$; since f is not constant, $f(x, \ldots, x) \succ_{E,\Sigma} x$, so $s \succ_{E,\Sigma} s$ by compatibility, a contradiction again.

We shall claim that, given the right ordering and selection function, we can make resolution start from clauses of certain forms, and only generate clauses of the same forms. These forms were the types **(1)**, **(2)** and **(3)** in Section 4.1. Here we shall use more general clauses.

Proposition 1 *Let Σ_0, Σ be two disjoint signatures, such that $\Sigma \cup \Sigma_0$ contains at least one non-constant function symbol. Assume E is a simple equational theory on Σ, and let $\succ_{E,\Sigma}$ be any computable strict, stable ordering $\succ_{E,\Sigma}$ compatible with E such that $f(x_1, \ldots, x_n) \succ_{E,\Sigma} x_i$ for every i, $1 \leq i \leq n$, for every $f \in \Sigma \cup \Sigma_0$. Let* sel *be the following selection function:*

- *If C is a Horn Σ-clause, possibly in disjunction with $+q$, $q \in \mathcal{Q}_0$, then:*
  - *If C contains a negative literal $-P(t)$ with t not a variable, then let* $\mathrm{sel}(C)$ *be $\{-P(t)\}$ (note that the fact that t is not a variable is well-defined modulo E, since E is simple, hence collapse-free);*
  - *Otherwise, if C can be written as $A \Leftarrow H, P_1(x), \ldots, P_m(x)$ where x is not free in A or in the body (i.e., conjunction of atoms) H and $m \geq 1$ (in other words, if there is a variable x free on the right of $\Leftarrow$ but not on the left), then let* $\mathrm{sel}(C)$ *be $\{-P_1(x), \ldots, -P_m(x)\}$.*
  - *Otherwise,* $\mathrm{sel}(C)$ *is empty.*
- *If C is a clause containing no function symbol from Σ, then:*
  - *if C contains a negative literal $-q$ with $q \in \mathcal{Q}$, then* $\mathrm{sel}(C) = \{-q\}$*;*
  - *otherwise, let* $\max(C)$ *be the set of maximal literals in C for $\succ$, then define* $\mathrm{sel}(C)$ *as the subset of those negative literals in* $\max(C)$*.*

For simplicity, say "resolution" for "ordered resolution with selection with ordering $\succ_{E,\Sigma}$ and selection function sel*".*

Given any set S of Horn clauses of one of the following types:

(4) *$C[\vee +q]$, where C is a block and $q \in \mathcal{Q}_0$;*
(5) *$C \vee -q_1 \vee \ldots \vee -q_m[\vee +q]$, where C is a complex Σ_0-clause, $m \geq 0$, and $q_1, \ldots, q_m, q \in \mathcal{Q}_0$;*
(6) *$C[\vee +q]$, where C is a non-trivial Σ-clause, and $q \in \mathcal{Q}_0$.*

then any clause obtained from S by resolution and eager ϵ-splitting on clauses not of type **(6)** *is again of one of the types* **(4)**, **(5)**, **(6)**.

Note that we restrict ϵ-splitting so as to not split clauses of type **(6)**. This is to avoid problems in the treatment of splitting in Σ-clauses.

PROOF. First, observe that: $(*)$ no non-variable Σ_0-term $s = f(s_1, \ldots, s_m)$ unifies with any non-variable Σ-term t modulo E. Indeed, assume otherwise: $E \models s\sigma \approx t\sigma$ for some substitution σ. Any term equal to s modulo E is of the form $f(\ldots)$, with $f \in \Sigma_0$, and since Σ is collapse-free, any term equal to $t\sigma$ modulo E must be of the form $g(\ldots)$ with $g \in \Sigma$. We conclude by observing that $f \neq g$, since Σ and Σ_0 are disjoint.

The case of clauses of type **(4)** and **(5)** has already been dealt with in Section 4.1, using the same selection function: clauses **(4)** and **(5)** resolve together to produce other clauses of type **(4)** or **(5)** only, up to ϵ-splitting.

It remains to examine the case where one clause of type **(6)** is resolved against some other clauses. There are two cases, depending on whether this clause is used as a main clause or as a side clause. Let the main clause be $C \vee -A_1 \vee \ldots \vee -A_m$, where $-A_1, \ldots, -A_m$ are the literals selected by sel, and let the side clauses be $A'_i \Leftarrow H_i$, $1 \leq i \leq m$.

- Let us assume some side clause $A'_i \Leftarrow H_i$ is of type **(6)**. Looking at the definition of sel, and remembering that no literal can be selected in a side clause, we realize that $A'_i \Leftarrow H_i$ must be of the form

  $$+P_i(g_i(t_{i1}, \ldots, t_{im_i})) \vee B_{i1}(x_1) \vee \ldots \vee B_{in}(x_{n_i})$$

  with $n_i \geq 2$, $g_i \in \Sigma$, and some negative blocks $B_1(x_1), \ldots, B_{n_i}(x_{n_i})$, and where $x_1, \ldots, x_{n_i}$ are free in $g_i(t_{i1}, \ldots, t_{im_i})$.

  This implies that the main clause cannot be of type **(5)**. Assume the contrary. $\succ_{E,\Sigma}$ and sel have been set up in such a way that the literals resolved upon, namely $A_1, \ldots, A_m$, would be maximal. Let A_i be written $P_i(t_i)$, with t_i a Σ_0-term. Observe that t_i cannot be a variable, say y: since $C \vee -A_1 \vee \ldots \vee -A_m$ is of type **(5)**, y would occur free in some non-variable term of the clause, contradicting maximality. So t_i is a non-variable Σ_0-term, hence A_i cannot unify with $g_i(t_{i1}, \ldots, t_{im_i})$ modulo E, by $(*)$ above.

  So the main clause is of type **(4)** or **(6)**.

  We claim that no other side clause $A'_j \Leftarrow H_j$, $j \neq i$, can be of type **(5)**. Indeed, assume otherwise. Since no literal is selected in side clauses, by definition of sel any side clause must contain no $-q$, $q \in \mathcal{Q}_0$, and the only maximal literal must be positive. So, any type **(5)** side clause must be of the form

$$P_j(f(t'_1, \ldots, t'_{n'})) \Leftarrow H_j$$

where $f \in \Sigma_0$. If the main clause is of type **(4)**, then by definition of sel, the selected literals $A_1, \ldots, A_m$ in it are of the form $-P_1(x), \ldots, -P_m(x)$ with the same variable x, so x unifies with both $g_i(t_{i1}, \ldots, t_{im_i})$ (ith side clause) and $f(t'_1, \ldots, t'_{n'})$ (jth), which is impossible by $(*)$. If the main clause is of type **(6)**, notice that sel was defined in such a way that the only possibility that it selects two or more atoms (required for i and j to be distinct) is in the second item of its definition, in which case the selected literals $A_1, \ldots,$ A_m are of the form $-P_1(x), \ldots, -P_m(x)$ with the same variable x, and the same argument goes through.

Since the main clause and all side clauses are of type **(4)** or **(6)**, it is clear now that the resolvent is of type **(6)**, if some symbol from Σ remains in it, or that it is a disjunction of blocks (and possibly of some $+q$, $q \in \mathcal{Q}_0$), which ϵ-splits into clauses of type **(4)** or **(5)**.

- Let us assume now that the main clause $C \vee -A_1 \vee \ldots \vee -A_m$ is of type **(6)**, but no side clause is. We have three cases, corresponding to the three cases defining sel:
  - $m = 1$, $A_1 = P(t)$, t is not a variable. In particular, t is a term that does not unify with any non-variable Σ_0-term, by $(*)$. In particular the side clause $A'_1 \Leftarrow H_1$ must be such that $A'_1 = P(x)$ for some variable x. Since no literal is selected in a side clause, H_1 contains no q literal, and also no literal that is maximal in $A'_1 \Leftarrow H_1$.

    We then claim that H_1 must be empty. Among all literals in H_1 where x is not free, there must be a maximal one, say $-Q(u)$. Now $Q(u)$ and x must be incomparable wrt. $\succ_{E,\Sigma}$: indeed, if $u \succ_{E,\Sigma} x$, since $\succ_{E,\Sigma}$ is stable and x is not free in u, we would have $u[x := u] \succ_{E,\Sigma} x[x := u]$, i.e., $u \succ_{E,\Sigma} u$, and similarly if $x \succ_{E,\Sigma} u$. But then $Q(u)$ would be maximal in $A'_1 \Leftarrow H_1$, where $A'_1 = P(x)$, a contradiction. So all literals in H_1 must contain x free; but since $\succ_{E,\Sigma}$ is a simplification ordering, the existence of such a literal in H_1 contradicts the maximality of $+P(x)$. So H_1 is empty.

    So $A'_1 \Leftarrow H_1$ is just the clause $+P(x)$, and the resolvent is C, which is of type **(6)** if C still contains a symbol from Σ, or splits into clauses of type **(4)** and **(5)** otherwise.
  - $A_1 = P_1(x), \ldots, A_m = P_m(x)$, and x is not free in C. Let σ be some member of csu $(A_1 \doteq A'_1, \ldots, A_m \doteq A'_m)$, and write A'_i as $P_i(t'_i)$, $1 \leq i \leq m$. Since σ unifies A_i with A'_i, it also unifies all the terms t'_i, $1 \leq i \leq m$.

    If some t'_i is a non-variable Σ-term, then none of the clauses $A'_j \Leftarrow H_j$, $1 \leq j \leq m$, can be of type **(5)**. Indeed, by the definition of sel, the only case a clause $A'_j \Leftarrow H_j$ of type **(5)** can be used as side clause (with no literal selected) is when $A'_j = P_j(t'_j)$ is such that t'_j is non-variable (a fact we have already used earlier). But if t'_j is non-variable, and is a Σ_0-term (since $A'_j \Leftarrow H_j$ is of type **(5)**), then it cannot unify with t'_i, by $(*)$.

Remember we have assumed that no side clause was of type **(6)**. So, if some t'_i is a non-variable Σ-term, then all side clauses are of type **(4)**, and by arguments similar to the previous case (the case $m = 1$, $A_1 = P(t)$, t not a variable), the generated resolvent is of type **(6)**, or splits into clauses of type **(4)** and **(5)**.

So we are left with the case where all side clauses are of type **(4)** or **(5)**. Without loss of generality, assume that the clauses of type **(4)** among the side clauses are $A'_i \Leftarrow H_i$, $1 \leq i \leq k$, and the clauses of type **(5)** are $A'_i \Leftarrow H_i$, $k+1 \leq i \leq m$. Remember that no literal is selected in $A'_i \Leftarrow H_i$. When $1 \leq i \leq k$, $A'_i = P_i(y)$ for some variable y, so by definition of sel, H_i must be empty. When $k+1 \leq i \leq m$, we use the fact that $A'_i \Leftarrow H_i$ is an automata clause for the first time. This special case of an automata clause with no literal selected must be of the form

$$P_i(f(x_{i1}, \ldots, x_{in})) \Leftarrow -B_{i1}(x_{i1}), \ldots, -B_{in}(x_{in})$$

where the variables $x_{i1}, \ldots, x_{in}$ are not necessarily distinct, and the $B_{ij}(x_{ij})$'s are negative blocks. The symbol f is the same for every i, because all A'_is must unify.

In the trivial case where $k = m$, i.e., when all side clauses are of type **(4)**, the resolvent is of type **(6)** or splits into clauses of type **(4)** and **(5)**. The interesting case is when $k < m$. Then there is a unique most general unifier σ of x, y, and the terms $f(x_{i1}, \ldots, x_{in})$, $k+1 \leq i \leq m$, and σ maps variables to variables. The resolvent is $\left(C \vee \bigvee_{i=k+1}^{m} \bigvee_{j=1}^{n} B_{ij}(x_{ij})\right) \sigma$, which is of type **(6)** if any symbol from Σ remains, or splits into clauses of type **(4)** and **(5)** otherwise.

- No literal is selected; by *(iv)*, $A_1, \ldots, A_m$ are maximal in $C \vee -A_1 \vee \ldots \vee -A_m$. More importantly, since no literal is selected in $C \vee -A_1 \vee \ldots \vee -A_m$, the definition of sel implies that the only negative literals in this clause are of the form $-P(x)$ with x a variable free at the left of the $\Leftarrow$ sign, i.e., that this clause is of the form

$$P(g(x_1, \ldots, x_n)) \Leftarrow -B_1(x_1), \ldots, -B_n(x_n)$$

with $g \in \Sigma$. But no negative literal is maximal here, contradiction. $\square$

The proof of Proposition 1 shows that, starting from clauses of type **(4)**, **(5)**, or **(6)**, resolution can be decomposed in two subclasses of the resolution rule: *white zone resolution*, where the premises are of type **(4)** or **(5)**, and *grey zone resolution*, where at least one premise is of type **(6)**. The latter has two variants: *light grey resolution* is when there is just one other premise, which is of type **(4)** or **(6)**; *dark grey resolution* is the one occurring in the next-to-last case of the proof of Proposition 1:

$$\frac{\begin{array}{l} P_i(x) \qquad (1 \leq i \leq k) \\ P_i(f(x_{i1}, \ldots, x_{in})) \Leftarrow -B_{i1}(x_{i1}), \ldots, -B_{in}(x_{in}) \\ \qquad (k+1 \leq i \leq m) \end{array} \qquad \begin{array}{l} C \vee -P_1(x) \vee \\ \ldots \vee -P_m(x) \end{array}}{\left(C \vee \bigvee_{i=k+1}^{m} \bigvee_{j=1}^{n} B_{ij}(x_{ij})\right)\sigma} \; (DGRes)$$

where $0 \leq k \leq m-1$, x is not free in C, C is of the form

$$P(g(y_1, \ldots, y_k)) \Leftarrow -B_1(y_1), \ldots, -B_k(y_k), -B_{k+1}(y_{k+1}), \ldots, -B_\ell(y_\ell)$$

with $g \in \Sigma$, and σ is the mgu of x and all $f(x_{i1}, \ldots, x_{in})$, $k+1 \leq i \leq m$.

Moreover, as said above, the only places where we get out of the grey zone, i.e. where we derive clauses that are not of type **(6)** by grey resolution, are when we generate splittable disjunctions of blocks (possibly in disjunction with $+q$, $q \in \mathcal{Q}_0$), which get ϵ-split immediately, yielding clauses of type **(4)** or **(5)**.

Let us abstract whatever may happen inside the grey zone by a unique rule: starting from a set S of clauses of type **(4)**, **(5)** or **(6)**, we guess which kind of clauses may be the fat dots terminating the grey zones. By the remark above, these fat dots are the *candidates*:

Definition 2 *A* candidate *is any Horn clause of the form $B_1(x_1) \vee \ldots \vee B_n(x_n)[\vee +q]$, where the x_is are pairwise distinct, $B_i(x_i)$ is a non-empty block for every i, $1 \leq i \leq n$, and $q \in \mathcal{Q}_0$.*

A grey oracle *is any function $\mathcal{O}$ mapping every set of clauses of type* **(4)**, **(5)**, *or* **(6)**, *to a set of candidates containing all those deducible by grey resolution.*

Corollary 3 *Let $\mathcal{O}$ be any grey oracle. Under the assumptions of Proposition 1, white resolution together with the* grey oracle rule*: to S add $\mathcal{O}(S)$, is complete for every set of clauses of type* **(4)**, **(5)**, *or* **(6)**. *In other words, for every set S_1 of clauses of type* **(4)**, **(5)**, *or* **(6)**, *if S_1 is unsatisfiable modulo E, then the empty clause can be derived from S_1 by white resolution and the grey oracle rule. Moreover, completeness is retained when removing tautologies, forward subsumed clauses, and ϵ-splitting of clauses not of type* **(6)**.

Please note that we did not require $\mathcal{O}$ to be *sound* (intuitively, such that only semantically valid consequences modulo E are produced by the oracle). This is not a problem here, since we only want a sufficient criterion for ensuring that S_1 is satisfiable: what we need is that if $\perp$ cannot be derived, then S_1 is satisfiable, and this is completeness. We also need $\mathcal{O}$ to be computable. We cannot require it to be sound as well, otherwise white resolution and the grey oracle rule would actually decide the class of all sets of clauses of type **(4)**, **(5)**, or **(6)**. Since this class contains the alternating two-way AC-tree automata of Goubault-Larrecq and Verma (2002), which is undecidable, this is hopeless.

Now finding a grey oracle is hard. In particular, dark grey resolution involves looking at clauses of all three types **(4)**, **(5)**, **(6)**. Remember we would like to find an oracle by just instantiating $\oplus$ as addition modulo n. This is impossible if clauses of type **(6)** have to be considered, since we would have to instantiate all function symbols from Σ_0 as well. We therefore strive to find replacement rules that do not need to look at clauses of type **(6)**.

Notice that the conclusion of dark grey resolution, $\left(C \vee \bigvee_{i=k+1}^{m} \bigvee_{j=1}^{n} B_{ij}(x_{ij})\right)\sigma$, is of the form $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$ for some distinct variables $x_1, \ldots, x_n$ not free in C and some negative ϵ-blocks $B_1(x_1), \ldots, B_n(x_n)$. Although this form of splitting was forbidden in Proposition 1, so as to keep some control on the shape of clauses of type **(6)**, we may imagine modifying dark grey resolution to a rule where the conclusion is ϵ-split right away, and the main premise is given in ϵ-split form. Let therefore *beige resolution* be

$$\frac{\begin{array}{ll} \begin{array}{l} P_i(x) \qquad (1 \leq i \leq k) \\ P_i(f(x_{i1}, \ldots, x_{in})) \Leftarrow -B_{i1}(x_{i1}), \ldots, -B_{in}(x_{in}) \\ \qquad (k+1 \leq i \leq m) \end{array} & \begin{array}{r} +q \vee -P_1(x) \vee \\ \ldots \vee -P_m(x) \end{array} \end{array}}{\begin{array}{c} +q \vee -q_1 \vee \ldots \vee -q_n \\ +q_j \vee B_j(x) \qquad (1 \leq j \leq n) \end{array}}$$

where $0 \leq k \leq m-1$, x is not free in C, C is of the form

$$P(g(y_1, \ldots, y_k)) \Leftarrow -B_1(y_1), \ldots, -B_k(y_k), -B_{k+1}(y_{k+1}), \ldots, -B_\ell(y_\ell)$$

with $g \in \Sigma$, σ is the mgu of x and all $f(x_{i1}, \ldots, x_{in})$, $k+1 \leq i \leq m$, and therefore $\left(C \vee \bigvee_{i=k+1}^{m} \bigvee_{j=1}^{n} B_{ij}(x_{ij})\right)\sigma$ can be written $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$, where $x_1, \ldots, x_n$ are distinct variables not free in C, and finally $q_j = \ulcorner B_j(x) \urcorner$. Just like splitting, this rule generates several clauses, $n+1$ to be precise.

For this to preserve completeness, we need to allow ϵ-splitting of clauses of type **(6)**, and to introduce the following *q-resolution* rule:

$$\frac{C \vee -q \qquad + q \vee -q_1 \vee \ldots \vee -q_m}{C \vee -q_1 \vee \ldots \vee -q_m} \; (qRes)$$

where C is a non-trivial Σ-clause. Once we do this, dark grey resolution can be simulated by first splitting the main clause $C \vee -P_1(x) \vee \ldots \vee -P_m(x)$ into $C \vee -q$ and $+q \vee -P_1(x) \vee \ldots \vee -P_m(x)$, where $q = \ulcorner -P_1(x) \vee \ldots \vee -P_m(x) \urcorner$. Then use beige resolution to derive $+q \vee -q_1 \vee \ldots \vee -q_n$ and $q_j \vee B_j(x)$, $1 \leq j \leq n$. By using q-resolution with $C \vee -q$, we get back $C \vee -P_1(x) \vee \ldots \vee -P_m(x)$, which is the conclusion of rule $(DGRes)$. Since beige resolution is just a special case of white resolution, we conclude:

Proposition 4 *Under the assumptions of Proposition 1, the rules of white resolution, light grey resolution, q-resolution and ϵ-splitting are complete.*

Unfortunately, Proposition 4 does not state that we may apply ϵ-splitting eagerly, which would be nicer for implementation purposes. This is repaired in the next proposition.

Proposition 5 *Let us say that a clause is of type* **(6')** *if and only if it is of the form $C \vee -q_1 \vee \ldots \vee -q_m[\vee + q]$, where C is a non-trivial Σ-clause, and $q_1, \ldots, q_m, q \in \mathcal{Q}_0$.*

Let ivory resolution *be the following modification of light grey resolution:*

$$\frac{+A' \vee C' \vee -q'_1 \vee \ldots \vee -q'_{m'} \qquad C \vee -q_1 \vee \ldots \vee -q_m \vee -A}{(C \vee C')\sigma \vee -q_1 \vee \ldots \vee -q_m \vee -q'_1 \vee \ldots \vee -q'_{m'}} \; (IvoryRes)$$

*where $\sigma \in \mathrm{csu}\ (A \doteq A')$; $+A'$ is maximal in $+A' \vee C'$ (*not *in $+A' \vee C' \vee -q'_1 \vee \ldots \vee -q'_{m'}$!) and no literal is selected in $+A' \vee C'$; and $-A$ is selected in $C \vee -A$, (*not *in $C \vee -q_1 \vee \ldots \vee -q_m \vee -A$!), or no literal is selected in $C \vee -A$ and $-A$ is maximal in $C \vee -A$; and moreover one of the premises is of type* **(6')***, and the other is of type* **(4)** *or* **(6')***.*

Then, under the assumptions of Proposition 1, the rules of white resolution, ivory resolution, q-resolution and eager ϵ-splitting are complete.

PROOF. Every clause can be written in a unique way $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$, where $x_1, \ldots, x_n$ are distinct variables that are not free in C, where $B_1(x_1), \ldots, B_n(x_n)$ are non-empty negative blocks, and n is maximal with these properties. By ϵ-splitting, we may derive $C \vee -q_1 \vee \ldots \vee -q_n$ and $+q_j \vee B_j(x_j)$ for every j, $1 \leq j \leq n$, from such a clause, where $q_j = \ulcorner B_j(x_j) \urcorner$. Call $C \vee -q_1 \vee \ldots \vee -q_n$ the *splinter* off $C \vee B_1(x_1), \ldots, B_n(x_n)$, and the clauses $+q_j \vee B_j(x_j)$ the *definitions* of q_j; the q_j symbols are the *splitting symbols*.

The plan of the proof is to show that if we can deduce any clause by white and light grey resolution, q-resolution and ϵ-splitting, then we can deduce its splinter by using ivory resolution instead of light grey resolution. It suffices to show that for any rule in the first set, the rules in the second set can be applied to the splinters of the premises to derive the splinter of the conclusion. This is obvious for white resolution, since ϵ-splitting was already applied eagerly in this case, and also for ϵ-splitting. Let us consider the other rules:

- Light grey resolution. Writing the premises in the form $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n)$, either we resolve upon a literal in C, or we resolve upon a literal in some $B_i(x_i)$. This means we have the following possibilities. First case:

$$\frac{+A' \vee C' \vee \bigvee_{i=1}^{m'} B'_i(x'_i) \qquad C \vee \bigvee_{i=1}^{m} B_i(x_i) \vee -A}{(C \vee C')\sigma \vee \bigvee_{i=1}^{m'} B'_i(x'_i) \vee \bigvee_{i=1}^{m} B_i(x_i)}$$

  By induction hypothesis, we have derived $C \vee -q_1 \vee \ldots \vee -q_n \vee -A$ and $+A' \vee C' \vee -q'_1 \vee \ldots \vee -q'_{n'}$, where $q_i = \ulcorner B_i(x_i) \urcorner$ and $q'_i = \ulcorner B'_i(x'_i) \urcorner$ for all i; we then infer $(C \vee C')\sigma \vee -q_1 \vee \ldots \vee -q_n \vee -q'_1 \vee \ldots \vee -q'_{n'}$ by ivory resolution. This clause is, or ϵ-splits into the splinter off the conclusion $(C \vee C')\sigma \vee \bigvee_{i=1}^{m'} B'_i(x'_i) \vee \bigvee_{i=1}^{m} B_i(x_i)$.

  Second case: $+A'$ on the left is inside $B'_i(x'_i)$ for some i. This is impossible since $B'_i(x'_i)$ is a negative block.

  Third case: $-A$ on the right is inside $B_i(x_i)$ for some i. By *(iv)* and the definition of sel, the only possibility is that the right premise is of type **(4)**, say $B(x) \vee -P(x)[\vee + q]$, where $B(x)$ is a block. Note that it is its own splinter. The left premise is $+A' \vee C' \vee \bigvee_{i=1}^{m'} B'_i(x'_i)$ where $A' = P(t')$ for some t'. Its splinter is $+P(t) \vee C' \vee -q'_1 \vee \ldots \vee -q'_{n'}$. By ivory resolution, we infer $B(t')[\vee + q] \vee C' \vee -q'_1 \vee \ldots \vee -q'_{n'}$, whose splinter is also that of the resolvent $B(t')[\vee + q] \vee \bigvee_{i=1}^{m'} B'_i(x'_i)$.
- q-resolution. The premises are of the form $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n) \vee -q$ and $+q \vee -q'_1 \vee \ldots \vee -q'_{n'}$, the resolvent is $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n) \vee -q'_1 \vee \ldots \vee -q'_{n'}$. The splinters off the premises are $C \vee -q_1 \vee \ldots \vee -q_n \vee -q$ ($q_i = \ulcorner B_i(x_i) \urcorner$ for all i) and $+q \vee -q'_1 \vee \ldots \vee -q'_{n'}$. By q-resolution, we infer $C \vee -q_1 \vee \ldots \vee -q_n \vee -q'_1 \vee \ldots \vee -q'_{n'}$, which is the splinter off $C \vee B_1(x_1) \vee \ldots \vee B_n(x_n) \vee -q'_1 \vee \ldots \vee -q'_{n'}$. $\square$

As before, we may infer a corresponding notion of oracle. Please note that, using the rules of Proposition 5, we now get out of grey zones, not by generating disjunctions of blocks $B_1(x_1) \vee \ldots \vee B_n(x_n)[\vee + q]$, but by producing their splinters $B_1(x_1) \vee -q_2 \vee \ldots \vee -q_n[\vee + q]$, plus definitions for splitting symbols. The latter include the symbols $q_2, \ldots, q_n$ but may contain others.

Definition 6 *A* light candidate *is either a clause $B(x) \vee -q_2 \vee \ldots \vee -q_n[\vee + q]$, where $B(x)$ is a block, or a definition $+\ulcorner B'(x) \urcorner \vee B'(x)$.*

It is accessible *from a set S of clauses of type* **(4)***,* **(5)***, or* **(6')***, if and only if it is derivable from S by ivory resolution and q-resolution alone. A* light grey oracle *is any function $\mathcal{O}$ mapping every set of clauses of type* **(4)***,* **(5)***, or* **(6')** *to a set of light candidates containing those that are accessible from S.*

Corollary 7 *Let $\mathcal{O}$ be any light grey oracle. Under the assumptions of Proposition 1, white resolution and the* light grey oracle rule*: to S add $\mathcal{O}(S)$, are complete for every set of clauses of type* **(4)***,* **(5)***, or* **(6')***. More precisely, for every set S_1 of clauses of type* **(4)***,* **(5)***, or* **(6')***, if S_1 is unsatisfiable modulo E, then the empty clause can be derived from S_1 by white resolution and the light grey oracle rule. Moreover, completeness is retained when removing tautologies, forward subsumed clauses, and eagerly ϵ-splitting clauses.*

This in particular applies to sets S_1 of flattened clauses obtained from sets S_0 describing Diffie-Hellman-like protocols (Section 3).

4.5 Finding a Light Grey Oracle

The main value of light grey oracles $\mathcal{O}$, justifying that we did not stop our analysis of Section 4.4 after Corollary 3, is that the function mapping S to $\mathcal{O}(S_{\mathbf{4},\mathbf{6'}})$ is also a light grey oracle, where $S_{\mathbf{4},\mathbf{6'}}$ is the subset of S containing just the clauses of type **(4)** and **(6')**. This is because the ivory resolution and q-resolution rules never apply to clauses of type **(5)**.

Without loss of generality, let us therefore assume in this section that S is a set of clauses of type **(4)** and **(6')**. Note that all terms in such clauses are built on Σ: no symbol from Σ_0 occurs anywhere in S. In the case of the theory AC, this means that the only function symbol in S is $\oplus$.

To design a *computable* light grey oracle, fix a finite model $\mathcal{M}$ of the equational theory E on signature Σ. In the AC case, this is an arbitrary finite semigroup. In practice, we choose $\mathbb{Z}/n\mathbb{Z}$ for some small value of n, typically $n = 3$.

Call $\mathcal{M}$*-instance* of a clause of type **(4)** and **(6')** any clause obtaining by replacing each variable by an element of $\mathcal{M}$, and simplifying. Formally, let ρ be an *environment*, i.e., a function mapping variables to elements of $\mathcal{M}$, let $[\![x]\!]\,\rho = \rho(x)$ and $[\![g(t_1, \ldots, t_n)]\!]\,\rho = \mathcal{M}_g([\![t_1]\!]\,\rho, \ldots, [\![t_n]\!]\,\rho)$, where $\mathcal{M}_g$ is some n-ary function (e.g., addition modulo n). Now let $[\![P(t)]\!]\,\rho$ be the formal atom $P([\![t]\!]\,\rho)$ for every atom $P(t)$; we call this an $\mathcal{M}$*-atom*. Similarly, define the $\mathcal{M}$*-literal* $[\![\pm P(t)]\!]\,\rho = \pm P([\![t]\!]\,\rho)$, and the $\mathcal{M}$*-clause* $[\![C]\!]\,\rho = \pm_1 [\![A_1]\!]\,\rho \vee \ldots \vee \pm_n [\![A_n]\!]\,\rho \vee \pm_{n+1} q_1 \vee \ldots \vee \pm_{n+m} q_m$, where $C = \pm_1 A_1 \vee \ldots \vee \pm_n A_n \vee \pm_{n+1} q_1 \vee \ldots \vee \pm_{n+m} q_m$. An $\mathcal{M}$*-instance* of clause C is any $\mathcal{M}$-clause $[\![C]\!]\,\rho$.

Our light grey oracle consists in simulating ivory resolution and q-resolution at the level of $\mathcal{M}$-instances.

To start off, we use an $\mathcal{M}$-*instantiation rule*: from any clause C in S, add all $\mathcal{M}$-instances of C to S. Then, we simulate ivory resolution and q-resolution by the *propositional* version of resolution on $\mathcal{M}$-instances (without any ordering or selection function):

Lemma 8 *Given a set ES of $\mathcal{M}$-clauses, let $Res_0(ES)$ be the set of all clauses deducible from ES by binary propositional resolution. Let S be any set of clauses of type* **(4)** *or* **(6')**, *$\mathcal{M}(S)$ the set of its $\mathcal{M}$-instances, and $Res(S)$ be the set of clauses deducible from S by ivory resolution and q-resolution. Then $\mathcal{M}(Res(S)) \subseteq Res_0(\mathcal{M}(S))$.*

This is immediate. Simulating ϵ-splitting at the level of $\mathcal{M}$-instances is more complicated. Notice that the set of $\mathcal{M}$-instances of $C \vee -P_1(x) \vee \ldots \vee -P_n(x)$, where x is not free in C, is exactly the set of $\mathcal{M}$-clauses $C_0 \vee -P_1(k) \vee \ldots \vee -P_n(k)$, where C_0 ranges over the $\mathcal{M}$-instances of C and k ranges over $\mathcal{M}$. So let us consider the following *abstract splitting* rule

$$\frac{\overbrace{C_0 \vee -P_1(k) \vee \ldots \vee -P_m(k)}^{k \in \mathcal{M}}}{C_0 \vee -q \qquad + q \vee -P_1(x) \vee \ldots \vee -P_m(x)} \ (ASplit)$$

where $q = \ulcorner -P_1(x) \vee \ldots \vee -P_m(x) \urcorner$, $m \geq 1$, and C_0 is not empty. This reads as follows: if, given a set S of $\mathcal{M}$-clauses and clauses of type **(4)**, **(5)**, or **(6')**, S contains *all* the $\mathcal{M}$-clauses $C_0 \vee -P_1(k) \vee \ldots \vee -P_m(k)$ $(k \in \mathcal{M})$, for some predicates $P_1, \ldots, P_m$, then add the $\mathcal{M}$-clause $C_0 \vee -q$ and the definition of the splitting symbol q to S.

Finally, we simulate the generation of fat dot clauses at the exit of grey zones (i.e., of light candidates) using the same idea: this is rule $(Exit)$ at the right, where $B(x)$ is any block.

$$\frac{\overbrace{B(k) \vee -q_2 \vee \ldots \vee -q_m [\vee + q]}^{k \in \mathcal{M}}}{B(x) \vee -q_2 \vee \ldots \vee -q_m [\vee + q]} \ (Exit)$$

We get the following immediate extension of Lemma 8.

Lemma 9 *Let S be any set of $\mathcal{M}$-clauses and clauses of type* **(4)**, **(5)**, *or* **(6')**. *Let $S_{|\mathcal{M}}$ the subset of $\mathcal{M}$-clauses.*

Let $Ded_0(S)$ be the set of all clauses and $\mathcal{M}$-clauses deducible by binary proposition resolution between $\mathcal{M}$-clauses and by the $(ASplit)$ and $(Exit)$ rules. Let $Ded(S) = Res(S_{|\mathcal{M}})$.

If $S_{|\mathcal{M}}$ contains $\mathcal{M}(S_{\mathbf{4,6'}})$, then $\mathcal{M}(Ded(S)) \subseteq Ded_0(S)$. Moreover, for every light candidate C accessible from $S_{\mathbf{4,6'}}$, C is in $Ded_0(S)$.

Proposition 10 *If $\mathcal{M}$ is a finite model of E on signature Σ, then the function $\mathcal{O}_{std}$ mapping any set of clauses of type* **(4)**, **(5)**, *or* **(6')** *to the set of light candidates in $Ded_0(S \cup \mathcal{M}(S))$ is a computable light grey oracle.*

4.6 Implementing the Light Grey Oracle

The only remaining challenge now is to implement the light grey oracle $\mathcal{O}_{std}$ of Proposition 10 efficiently.

Let us consider the easier rule $(Exit)$ first. To handle it, we use a table Tab_{Exit}, mapping clauses to natural numbers. We maintain the invariant that for each clause C of the form $\pm_1 P_1(x) \vee \ldots \vee \pm_m P_m(x) \vee -q_2 \vee \ldots \vee -q_n [\vee + q]$ (i.e., those that may be the conclusion of $(Exit)$), $\mathsf{Tab}_{Exit}(C)$ is the number of $\mathcal{M}$-instances of C that are present in the current clause set S. Note here that C is not limited to be a clause in S: C is arbitrary.

This invariant is maintained as follows. Initially, Tab_{Exit} is empty, i.e., it maps every clause to 0. For each newly-generated $\mathcal{M}$-clause in the current clause set S (by $\mathcal{M}$-instantiation, or by binary propositional resolution), first check that it is of the form $\pm_1 P_1(k) \vee \ldots \vee \pm_m P_m(k) \vee -q_2 \vee \ldots \vee -q_n [\vee + q]$, with the *same* value k as argument to all predicates $P_1, \ldots, P_m$. If it is of this form, then generate the *skeleton* clause $C = \pm_1 P_1(x) \vee \ldots \vee \pm_m P_m(x) \vee -q_2 \vee \ldots \vee -q_n [\vee + q]$, and add 1 to $\mathsf{Tab}_{Exit}(C)$. Notice that this is only correct provided that $\pm_1 P_1(k) \vee \ldots \vee \pm_m P_m(k) \vee -q_2 \vee \ldots \vee -q_n [\vee + q]$ is indeed new, i.e., was not already present in S. This is handled by the subsumption engine.

Now, $(Exit)$ is implemented by producing the clause C as soon as $\mathsf{Tab}_{Exit}(C)$ reaches n, the cardinality of $\mathcal{M}$: $(Exit)$ is as easy as using a table and counting.

The case of $(ASplit)$ is slightly trickier. As above, create a table Tab_{ASplit} mapping pairs $(C_0, -P_1(x) \vee \ldots \vee -P_m(x))$ of a non-empty $\mathcal{M}$-clause C_0 and a non-empty negative block $-P_1(x) \vee \ldots \vee -P_m(x)$ to natural numbers. The invariant is that $\mathsf{Tab}_{ASplit}(C_0, -P_1(x) \vee \ldots \vee -P_m(x))$ is the number of $\mathcal{M}$-clauses of the form $C_0 \vee -P_1(k) \vee \ldots \vee -P_m(k)$, $k \in \mathcal{M}$, in the current clause set S. Initially, Tab_{ASplit} is empty. For each newly-generated $\mathcal{M}$-clause C in the current clause set S, if C is not empty then it contains some literal $-P_1(k)$, $k \in \mathcal{M}$. For every such k, write C as $C_0 \vee -P_1(k) \vee \ldots \vee -P_m(k)$, where $m \geq 1$ is maximal. Then, for every pair (B, B') of subsets of $\{P_1, \ldots, P_m\}$ such that B is not empty, not both B' and C_0 are empty, and $B \cup B' = \{P_1, \ldots, P_m\}$, then add one to the entry $\mathsf{Tab}_{ASplit}(C_0 \vee B'(k), B(x))$.

This is sound: the only clauses of the form $(C'_0, -P'_1(x) \vee \ldots \vee -P'_{m'}(x))$ with $m' \geq 1$ and C'_0 not empty such that $C'_0 \vee -P'_1(k) \vee \ldots \vee -P'_{m'}(k)$ is C for some k are such that k occurs as argument of some predicate in C. Then

$B = \{P'_1, \ldots, P'_{m'}\}$ is a non-empty subset of $\{P_1, \ldots, P_m\}$, and C'_0 is of the form $C_0 \vee B'(k)$ for some $B' \subseteq \{P_1, \ldots, P_m\}$ such that $B \cup B' = \{P_1, \ldots, P_m\}$, namely B' is set of all those predicates applied to k in C'_0).

For example, starting from $C = -P(2) \vee -Q(1) \vee -R(1)$, we get $C_0 = -P(2)$ if we choose $k = 1$, then we add one to the entry for the pair $(-P(2), -Q(x) \vee -R(x))$. This is the most obvious case, and we get back C by taking the disjunction of $-P(2)$ with the $\mathcal{M}$-instance obtained from $-Q(x) \vee -R(x)$ by taking $x = 1$. We also add one, e.g., to the entry $(-P(2) \vee -Q(1), -R(x))$, corresponding to the fact that C is also the disjunction of $-P(2) \vee -Q(1)$ with the $\mathcal{M}$-instance $-R(1)$ of $-R(x)$. We also add one to the entry for $(-P(2) \vee -Q(1), -Q(x) \vee -R(x))$: the corresponding $\mathcal{M}$-instance is $-P(2) \vee -Q(1) \vee -Q(1) \vee -R(1)$, which is C again, since the two $-Q(1)$'s merge.

Finally, when $\mathsf{Tab}_{ASplit}(C_0, -P_1(x) \vee \ldots \vee -P_m(x))$ reaches n, we just add the corresponding conclusions $C_0 \vee -q$ and $+q \vee -P_1(x) \vee \ldots \vee -P_m(x))$ of $(ASplit)$ to the current clause set S, where $q = \ulcorner -P_1(x) \vee \ldots \vee -P_m(x) \urcorner$.

This was implemented as part of the MOP modular platform for automated theorem proving (Roger, 2003).

5 Running MOP on the Group Diffie-Hellman Protocol

We have already seen how to formalize the group Diffie-Hellman protocol with a fixed number of principals in Section 3.2, using an associative-commutative primitive $\oplus$, a unary function symbol e and a constant $\mathbf{0}$. Running MOP with a light grey oracle on $\mathbb{Z}/3\mathbb{Z}$, we get the statistics shown in Figure 2 for the case with two principals, in Figure 3 for three principals, in Figure 4 for four principals. Experiments were conducted on a 1.4 GHz Intel-class machine with 512 Mb memory and 256 Kb cache memory, running Linux 2.4.3-20. MOP is written in OCaml 3. Note that the code was not especially optimized; notably, all hash tables are actually implemented as association lists, and subsumption tests do not use any indexing technique.

In the copycat and Dolev-Yao models, we retrieve the standard attack (Millen and Denker, 2002). More interestingly, MOP shows that IKA.1 is *secure* in the pure eavesdropper model, which was after all the only model it was designed for (Steiner et al., 2000). As far as we know, this is the first completely automated proof of this fact. (The proof by Pereira and Quisquater (2001) is by hand.)

This verifies IKA.1 assuming that $\oplus$ is associative and commutative, but it would be more realistic to consider that $\oplus$ is an Abelian group law. So take Σ to consist of $\oplus$ and another, unary symbol $\ominus$ (opposite), and E be the

Intruder model	Pure eavesdropper	Copycat	Dolev-Yao
Result	protocol *safe*	$\perp$ inferred (may be unsafe)	$\perp$ inferred (may be unsafe)
Initial clauses	118	122	162
Created clauses	1 364	1 530	2 313
... by resolution	570	544	1 244
... by splitting	28	23	41
... by $\mathcal{M}$-instantiation	729	927	990
... by (*Exit*)	13	12	14
... by (*ASplit*)	24	24	24
Time (s.)	1.31	1.64	2.57
Memory (Kb)	2 992	3 240	3 244

Fig. 2. Verifying IKA.1 with 2 principals

Intruder model	Pure eavesdropper	Copycat	Dolev-Yao
Result	protocol *safe*	$\perp$ inferred (may be unsafe)	$\perp$ inferred (may be unsafe)
Initial clauses	640	655	805
Created clauses	9 318	4 296	3 075
... by resolution	3 848	915	452
... by splitting	180	87	75
... by $\mathcal{M}$-instantiation	4 905	2 922	2 178
... by (*Exit*)	109	96	94
... by (*ASplit*)	276	276	276
Time (s.)	69.80	11.72	7.21
Memory (Kb)	6 416	5 024	4 988

Fig. 3. Verifying IKA.1 with 3 principals

theory $\mathrm{ACD}_1 = \mathrm{AC} \cup \{\ominus(x \oplus y) \approx (\ominus x) \oplus (\ominus y)\}$. This is again a simple equational theory: define $||x|| = 1$ for every variable x, $||s \oplus t|| = ||s|| + ||t||$, $|| \ominus s|| = 2||s||$, and $||f(t_1, \ldots, t_n)|| = 1 + ||t_1|| + \ldots + ||t_n||$ for all other function symbols f, then the ordering $s \succ_{E,\Sigma} t$ if and only if $||s|| > ||t||$ is a compatible simplification ordering. So our construction applies again. We need to add intruder clauses of the form *(a)* $I_C(\ominus x) \Leftarrow I_C(x)$, and clauses *(b)* $ch_C(\ominus x) \Leftarrow ch_C(x)$ in the copycat and Dolev-Yao models. We also need to

Intruder model	Pure eavesdropper	Copycat	Dolev-Yao
Result	protocol *safe*	$\perp$ inferred (may be unsafe)	$\perp$ inferred (may be unsafe)
Initial clauses	2 020	2 060	2 460
Created clauses	48 593	11 568	7 230
... by resolution	11 049	2 374	339
... by splitting	444	202	156
... by $\mathcal{M}$-instantiation	35 937	7 863	5 610
... by (*Exit*)	323	289	283
... by (*ASplit*)	840	840	842
Time (s.)	4 438	97.63	58.65
Memory (Kb)	36 252	10 420	9 352

Fig. 4. Verifying IKA.1 with 4 principals

deal with the collapsing axioms $x \oplus \mathbf{0} \approx x$ and $\ominus(\ominus x) \approx x$, and the axiom $x \oplus (\ominus x) \approx \mathbf{0}$. This can be handled by the Meadows trick (see Section 2.1): for every predicate symbol P, add clauses $P(x \oplus \mathbf{0}) \Leftarrow P(x)$ and $P(x) \Leftarrow P(x \oplus \mathbf{0})$, $P(\ominus(\ominus x)) \Leftarrow P(x)$ and $P(x) \Leftarrow P(\ominus(\ominus x))$, $P(x \oplus (\ominus x)) \Leftarrow P(\mathbf{0})$ and $P(\mathbf{0}) \Leftarrow P(x \oplus (\ominus x))$. Although they are not flattened, all these are clauses of type **(6')**, so our construction still applies. The nice thing is that all of these new clauses $\mathcal{M}$-instantiate to tautologies when $\mathcal{M}$ is an Abelian group, and our procedure *removes* tautologies. $\mathbb{Z}/n\mathbb{Z}$ is an Abelian group: so we just need to add clauses *(a)* and *(b)*, and no clause arising from Meadows' trick at all.

Running MOP with this slight modification, we find that IKA.1 is indeed again *secure* under the assumption that $\oplus$ is an Abelian group law. This takes 1.32 seconds for 2 participants (122 initial clauses, 1 376 created clauses, of which 570 by resolution, 28 by splitting, 741 by $\mathcal{M}$-instantiation, 13 by (*Exit*) and 24 by (*ASplit*)); 75.45 seconds for 3 participants (655 initial clauses, 9 575 created clauses of which 3 772 by resolution, 183 by splitting, 5 235 $\mathcal{M}$-instantiations, 109 by (*Exit*), 276 by (*ASplit*)); 5 632 seconds for 4 participants (2 060 initial clauses, 52 190 created clauses of which 10 943 by resolution, 448 by splitting, 39 636 $\mathcal{M}$-instantiations, 323 by (*Exit*), 840 by (*ASplit*)) In fact, this can also be shown using $n = 2$ instead of $n = 3$. We then obtain that IKA.1 is secure for 4 participants in 439.18 seconds (2 060 initial clauses, 19 563 created clauses, of which 8 932 by resolution, 448 by splitting, 9 100 $\mathcal{M}$-instantiations, 283 by (*Exit*), 800 by (*ASplit*)).

Currently, verifying IKA.1 with 4 principals seems to be an upper limit. Notice that the number of clauses describing IKA.1 with k principals grows as an exponential of k already. One particularly good news is that, although the light grey oracle rules may seem unwieldy, they are efficient enough in practice, and the number of clauses generated by the light grey oracle is in fact small. But n must be kept low: with $n = 3$, already 45-75% of all generated clauses are $\mathbb{Z}/n\mathbb{Z}$-instances.

6 Conclusion

We have shown how cryptographic protocols using Diffie-Hellman primitives, i.e., modular exponentiation on a fixed generator, can be encoded in Horn clauses modulo associativity and commutativity. In order to obtain a sufficient criterion of security, we have designed a complete (but not sound in general) resolution procedure for a class of flattened clauses modulo simple equational theories, including associativity-commutativity. This provides a sound abstraction of the initial protocol. More than that: our techniques provide a sufficient criterion of satisfiability for the undecidable class of alternating two-way AC-tree automata (Goubault-Larrecq and Verma, 2002).

The second author has implementated this algorithm in the MOP modular platform for automated proving. While the number of clauses representing the protocol already grows exponentially fast with the number of participants, this works well enough for small numbers of participants. The core of our resolution procedure consists in *light grey oracle* rules, which, together with white resolution, are complete; this was the hard technical point of the paper. Finding a light grey oracle, although seemingly hard, turned out to be feasible using fairly simple hashing and counting mechanisms. While our light grey oracle might seem unwieldy as far as complexity is concerned, experience proves otherwise: it is efficient enough and does not generate that many clauses.

As a final note, we stress that we have obtained the first fully automated proof of security of the IKA.1 initial key agreement protocol in the pure eavesdropper model, the model it was designed to work in (Steiner et al., 2000).

References

Amadio, R. and Charatonik, W. (2002). On name generation and set-based analysis in the Dolev-Yao model. In *13th International Conference on Concurrency Theory (CONCUR)*, pages 499–514. Springer-Verlag LNCS 2421.

Bachmair, L. and Ganzinger, H. (2001). Resolution theorem proving. In Robinson and Voronkov (2001), chapter 2, pages 19–99.

Bachmair, L., Ganzinger, H., and Waldmann, U. (1993). Set constraints are the monadic class. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83. IEEE Computer Society Press.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society Press.

Chevalier, Y., Küsters, R., Rusinowitch, M., and Turuani, M. (2003). Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In Pandya, P. and Radhakrishnan, J., editors, *23rd Intl. Conf. Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, Mumbai, India. Springer Verlag LNCS.

Comon, H., Cortier, V., and Mitchell, J. (2001). Tree automata with one memory, set constraints and ping-pong protocols. In *28th International Conference on Automata, Languages and Programming (ICALP)*, pages 682–693. Springer-Verlag LNCS 2076.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`.

Comon, H. and Shmatikov, V. (2002). Is it possible to decide whether a cryptographic protocol is secure or not ? In Goubault-Larrecq (2002b), pages 3–13.

Comon-Lundh, H. and Cortier, V. (2003). New decidability results for fragments of first-order logic and application to cryptographic protocols. In Nieuwenhuis (2003), pages 148–164.

Cousot, P. and Cousot, R. (1992). Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179. Correct version at `http://www.dmi.ens.fr/~cousot/COUSOTpapers/JLP92.shtml`.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208.

Fagès, F. (1984). Associative-commutative unification. In Shostak, R. E., editor, *7th Int. Conf. on Automated Deduction (CADE)*, pages 194–208, Napa Valley, California, USA. Springer Verlag LNCS 170.

Fermüller, C., Leitsch, A., Hustadt, U., and Tammet, T. (2001). *Resolution Decision Procedures*, chapter 25, pages 1791–1849. Volume II of Robinson and Voronkov (2001).

Frühwirth, T., Shapiro, E., Vardi, M. Y., and Yardeni, E. (1991). Logic programs as types for logic programs. In *6th Symposium Logic in Computer Science (LICS)*, pages 300–309. IEEE Computer Society Press.

Genet, T. and Klay, F. (2000). Rewriting for cryptographic protocol verification. In *17th Int. Conf. on Automated Deduction (CADE)*, pages 271–290, Pittsburgh, PA. Springer Verlag LNAI 1831.

Goldwasser, S. and Bellare, M. (1999). Lecture notes on cryptography. `http://www-cse.ucsd.edu/~mihir/papers/gb.ps.gz`.

Goré, R., Leitsch, A., and Nipkow, T., editors (2001). *1st Intl. Joint Conf. Automated Reasoning (IJCAR)*, Siena, Italy. Springer Verlag LNAI 2083.

Goubault-Larrecq, J. (2000). A method for automatic cryptographic protocol verification (extended abstract). In *Int. Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPPTA)*, pages 977–984. Springer Verlag LNCS 1800.

Goubault-Larrecq, J. (2002a). Higher-order positive set constraints. In *16th Int. Workshop Computer Science Logic (CSL)*, pages 473–489, Edinburgh, Scotland. Springer Verlag LNCS 2471.

Goubault-Larrecq, J., editor (2002b). *Journal of Telecommunications and Information Technology. Special Issue on Models and Methods for Cryptographic Protocol Verification*, volume 4. Instytut Łącsnośći (Institute of Telecommunications), Warsaw, Poland.

Goubault-Larrecq, J. (2002c). Vérification de protocoles cryptographiques : la logique à la rescousse ! In Goubault-Larrecq, J., editor, *Actes du 1er workshop international sur la sécurité des communications sur Internet (SECI'02)*, pages 119–152. INRIA, collection didactique.

Goubault-Larrecq, J. (2003). Résolution ordonnée avec sélection et classes décidables de la logique du premier ordre. Lecture notes for the course "démonstration automatique et vérification de protocoles cryptographiques" (with Hubert Comon-Lundh), DEA "programmation". 70 pages, `http://www.lsv.ens-cachan.fr/~goubault/SOresol.ps`.

Goubault-Larrecq, J. (2004). Une fois qu'on n'a pas trouvé de preuve, comment le faire comprendre à un assistant de preuve? In Ménissier-Morain, V., editor, *Actes des 12èmes Journées Francophones des Langages Applicatifs (JFLA'04)*. INRIA, collection didactique.

Goubault-Larrecq, J. and Verma, K. N. (2002). Alternating two-way AC-tree automata. Research Report LSV-02-11, LSV. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rr-lsv-2002-11.rr.ps`.

Joyner Jr., W. H. (1976). Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417.

Kapur, D., Narendran, P., and Wang, L. (2003). An E-unification algorithm for analyzing protocols that use modular exponentiation. In Nieuwenhuis (2003), pages 165–179.

Lowe, G. (1996). An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133.

Meadows, C. (1996). The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131.

Millen, J. and Denker, G. (2002). CAPSL and MuCAPSL. In Goubault-Larrecq (2002b), pages 15–25.

Monniaux, D. (1999). Abstracting cryptographic protocols with tree automata. In *6th International Static Analysis Symposium (SAS'99)*, pages 149–163. Springer-Verlag LNCS 1694.

Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

Nieuwenhuis, R., editor (2003). *14th Intl. Conf. Rewriting Techniques and Applications (RTA)*, Valencia, Spain. Springer Verlag LNCS 2706.

Pereira, O. and Quisquater, J.-J. (2001). A security analysis of the cliques protocols suites. In *14th IEEE Computer Security Foundations Workshop*, pages 73–81.

Plotkin, G. (1972). Building in equational theories. *Machine Intelligence*, 7:73–90.

Riazanov, A. and Voronkov, A. (2001). Vampire 1.1 (system description). In Goré et al. (2001), pages 376–380.

Robinson, J. A. and Voronkov, A., editors (2001). *Handbook of Automated Reasoning*. North-Holland.

Roger, M. (2003). *Raffinements de la résolution et vérification de protocoles cryptographiques*. PhD thesis, ENS de Cachan.

Selinger, P. (2001). Models for an adversary-centric protocol logic. *Electronic Notes in Theoretical Computer Science*, 55(1):73–87. 1st Workshop on Logical Aspects of Cryptographic Protocol Verification (LACPV), J. Goubault-Larrecq, ed.

Steiner, M., Tsudik, G., and Waidner, M. (2000). Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780.

Stickel, M. E. (1981). A unification algorithm for associative-commutative functions. *Journal of the ACM*, 28:423–434.

Verma, K. N. (2003a). *Automates d'arbres bidirectionnels modulo théories équationnelles*. PhD thesis, École Normale Supérieure de Cachan.

Verma, K. N. (2003b). On closure under complementation of equational tree automata for theories extending AC. In *10th Intl. Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 183–197, Almaty, Kazakhstan. Springer Verlag LNCS 2850.

Verma, K. N. (2003c). Two-way equational tree automata for AC-like theories: Decidability and closure properties. In Nieuwenhuis (2003), pages 180–196.

Voronkov, A. (2001). Algorithms, datastructures, and other issues in efficient automated deduction. In Goré et al. (2001), pages 13–28.

Weidenbach, C. (1999). Towards an automatic analysis of security protocols. In Ganzinger, H., editor, *16th Int. Conf. on Automated Deduction (CADE)*, pages 378–382. Springer-Verlag LNAI 1632.

Weidenbach, C. (2001). *Combining Superposition, Sorts and Splitting*, chapter 27, pages 1965–2013. Volume II of Robinson and Voronkov (2001).