Flat and One-Variable Clauses: Complexity of Verifying Cryptographic Protocols with Single Blind Copying

Helmut Seidl Kumar Neeraj Verma

Institut für Informatik, TU München, Germany
{seidl,verma}@in.tum.de

Abstract. Cryptographic protocols with single blind copying were defined and modeled by Comon and Cortier using the new class $\mathcal{C}$ of first order clauses, which extends the Skolem class. They showed its satisfiability problem to be in 3-DEXPTIME. We improve this result by showing that satisfiability for this class is NEXPTIME-complete, using new resolution techniques. We show satisfiability to be DEXPTIME-complete if clauses are Horn, which is what is required for modeling cryptographic protocols. While translation to Horn clauses only gives a DEXPTIME upper bound for the secrecy problem for these protocols, we further show that this secrecy problem is actually DEXPTIME-complete.

1 Introduction

Several researchers have pursued modeling of cryptographic protocols using first order clauses [3, 6, 15] and related formalisms like tree automata and set constraints[5, 11, 12]. While protocol insecurity is NP-complete in case of a bounded number of sessions [14], this is helpful only for detecting some attacks. For certifying protocols, the number of sessions cannot be bounded, although we may use other safe abstractions. The approach using first order clauses is particularly useful for this class of problems. A common safe abstraction is to allow a bounded number of nonces, i.e. random numbers, to be used in infinitely many sessions. Security however still remains undecidable [5]. Hence further restrictions are necessary to obtain decidability.

In this direction, Comon and Cortier [6, 8] proposed the notion of protocols with single blind copying. Intuitively this restriction means that agents are allowed to copy at most one piece of data blindly in any protocol step, a restriction satisfied by most protocols in the literature. Comon and Cortier modeled the secrecy problem for these protocols using the new class $\mathcal{C}$ of first order clauses, which extends the Skolem class, and showed satisfiability for $\mathcal{C}$ to be decidable [6] in 3-DEXPTIME [8]. The NEXPTIME lower bound is easy. We show in this paper that satisfiability of this class is in NEXPTIME, thus NEXPTIME-complete. If clauses are restricted to be Horn, which suffices for modeling of cryptographic protocols, we show that satisfiability is DEXPTIME-complete (again the lower bound is easy). While translation to clauses only gives a DEXPTIME upper bound for the secrecy problem for this class of protocols, we further show that the secrecy problem for these protocols is also DEXPTIME-complete.

For proving our upper bounds, we introduce several variants of standard ordered resolution with selection and splitting [2]. Notably we consider resolution as consisting of instantiation of clauses, and of generation of propositional implications. This is

in the style of Ganzinger and Korovin [10], but we enhance this approach, and generate interesting implications to obtain optimal complexity. More precisely, while the approach of Ganzinger and Korovin [10] has a single phase of instantiation followed by propositional satisfiability checking, we generate certain interesting propositional implications, instantiate them, and iterate the process. We further show how this technique can be employed also in presence of rules for replacement of literals in clauses, which obey some ordering constraints. To deal with the notion of single blind copying we show how terms containing a single variable can be decomposed into simple terms whose unifiers are of very simple forms. As byproducts, we obtain optimal complexity for several subclasses of $\mathcal{C}$, involving so called *flat* and *one-variable* clauses.

Outline: We start in Section 2 by recalling basic notions about first order logic and resolution refinements. In Section 3 we introduce cryptographic protocols with single blind copying, discuss their modeling using the class $\mathcal{C}$ of first order clauses, and show that their secrecy problem is DEXPTIME-hard. To decide the class $\mathcal{C}$ we start with the subclass of one-variable clauses in Section 4 and show its satisfiability to be DEXPTIME-complete. Satisfiability of the fragment of $\mathcal{C}$ involving flat clauses is shown to NEXPTIME-complete in Section 5. In Section 6, the techniques from the two cases are combined with further ideas to show that satisfiability for $\mathcal{C}$ is NEXPTIME-complete. In Section 7 we adapt this proof to show that satisfiability for the Horn fragment of $\mathcal{C}$ is DEXPTIME-complete.

2 Resolution

We recall standard notions from first order logic. Fix a signature Σ of function symbols each with a given arity, and containing at least one zero-ary symbol. Let r be the maximal arity of function symbols in Σ. Fix a set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots\}$ of variables. Note that $\mathbf{x}_1, \mathbf{x}_2, \ldots$ (in bold face) are the actual elements of $\mathbf{X}$, where as $x, y, z, x_1, y_1, \ldots$ are used to represent arbitrary elements of $\mathbf{X}$. The set $T_\Sigma(\mathbf{X})$ of terms built from Σ and $\mathbf{X}$ is defined as usual. T_Σ is the set of *ground terms*, i.e. those not containing any variables. *Atoms* A are of the form $P(t_1, \ldots, t_n)$ where P is an n-ary predicate and t_i's are terms. *Literals* L are either positive literals $+A$ (or simply A) or negative literals $-A$, where A is an atom. $-(-A)$ is another notation for A. $\pm$ denotes $+$ or $-$ and $\mp$ denotes the opposite sign (and similarly for notations $\pm', \mp', \ldots$). A *clause* is a finite set of literals. A *negative clause* is one which contains only negative literals. If M is any term, literal or clause then the set $\mathsf{fv}(M)$ of variables occurring in them is defined as usual. If C_1 and C_2 are clauses then $C_1 \vee C_2$ denotes $C_1 \cup C_2$. $C \vee \{L\}$ is written as $C \vee L$ (In this notation, we allow the possibility of $L \in C$). If $C_1, \ldots, C_n$ are clauses such that $\mathsf{fv}(C_i) \cap \mathsf{fv}(C_j) = \emptyset$ for $i \neq j$, and if C_i is non-empty for $i \geq 2$, then the clause $C_1 \vee \ldots \vee C_n$ is also written as $C_1 \sqcup \ldots \sqcup C_n$ to emphasize this property. *Ground literals and clauses* are ones not containing variables. A term, literal or clause is *trivial* if it contains no function symbols. A substitution is a function $\sigma : \mathbf{X} \to T_\Sigma(\mathbf{X})$. *Ground substitutions* map every variable to a ground term. We write $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ to say that $x_i\sigma = t_i$ for $1 \leq i \leq n$ and $x\sigma = x$ for $x \notin \{x_1, \ldots, x_n\}$. If M is a term, literal, clause, substitution or set of such objects, then the effect $M\sigma$ of applying σ to M is defined as usual. *Renamings* are bijections $\sigma : \mathbf{X} \to \mathbf{X}$. If M is a term, literal,

clause or substitution, then a renaming of M is of the form $M\sigma$ for some renaming σ, and an instance of M is of the form $M\sigma$ for some substitution σ. If M and N are terms or literals then a *unifier* of M and N is a substitution σ such that $M\sigma = N\sigma$. If such a unifier exists then there is also a *most general unifier (mgu)*, i.e. a unifier σ such that for every unifier σ' of M and N, there is some σ'' such that $\sigma' = \sigma\sigma''$. Most general unifiers are unique upto renaming: if σ_1 and σ_2 are two mgus of M and N then σ_1 is a renaming of σ_2. Hence we may use the notation $mgu(M, N)$ to denote one of them. We write $M[x_1, \ldots, x_n]$ to say that $\mathsf{fv}(M) \subseteq \{x_1, \ldots, x_n\}$. If $t_1, \ldots, t_n$ are terms then $M[t_1, \ldots, t_n]$ denotes $M\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. If N is a set of terms them $M[N] = \{M[t_1, \ldots, t_n] \mid t_1, \ldots, t_n \in N\}$. If M is a set of terms, atoms, literals or clauses them $M[N] = \bigcup_{m \in M} m[N]$. A *Herbrand interpretation* $\mathcal{H}$ is a set of ground atoms. A clause C is *satisfied* in $\mathcal{H}$ if for every ground substitution σ, either $A \in \mathcal{H}$ for some $A \in C\sigma$, or $A \notin \mathcal{H}$ for some $-A \in C\sigma$. A set S of clauses is satisfied in $\mathcal{H}$ if every clause of S is satisfied in $\mathcal{H}$. If such a $\mathcal{H}$ exists then S is *satisfiable*, and $\mathcal{H}$ is a *Herbrand model* of S. A *Horn clause* is one containing at most one positive literal. If a set of Horn clauses is satisfiable then it has a least Herbrand model wrt the subset ordering.

Resolution and its refinements are well known methods for testing satisfiability of clauses. Given a strict partial order $<$ on atoms, a literal $\pm A$ is *maximal* in a clause C if there is no literal $\pm' B \in C$ with $A < B$. *Binary ordered resolution* and *ordered factorization* wrt ordering $<$ are defined by the following two rules respectively:

$$\frac{C_1 \vee A \quad -B \vee C_2}{C_1\sigma \vee C_2\sigma} \qquad \frac{C_1 \vee \pm A \vee \pm B}{C_1\sigma \vee A\sigma}$$

where $\sigma = mgu(A, B)$ in both rules, A and B are maximal in the left and right premises respectively of the first rule, and A and B are both maximal in the premise of the second rule. We rename the premises of the first rule before resolution so that they don't share variables. The ordering $<$ is *stable* if: whenever $A_1 < A_2$ then $A_1\sigma < A_2\sigma$ for all substitutions σ. We write $S \Rightarrow_< S \cup \{C\}$ to say that C is obtained by one application of the binary ordered resolution or binary factorization rule on clauses in S (the subscript denotes the ordering used).

Another resolution rule is *splitting*. This can be described using *tableaux*. A *tableau* is of the form $S_1 \mid \ldots \mid S_n$, where $n \geq 0$ and each S_i, called a *branch* of the tableau, is a set of clauses (the $\mid$ operator is associative and commutative). A tableau is *satisfiable* if at least one of its branches is satisfiable. The tableau is called *closed* if each S_i contains the empty clause, denoted $\Box$. The *splitting* step on tableaux is defined by the rule: $\mathcal{T} \mid S \rightarrow_{spl} \mathcal{T} \mid (S \backslash \{C_1 \sqcup C_2\}) \cup \{C_1\} \mid (S \backslash \{C_1 \sqcup C_2\}) \cup \{C_2\}$ whenever $C_1 \sqcup C_2 \in S$ and C_1 and C_2 are non-empty. C_1 and C_2 are called *components* of the clause $C_1 \sqcup C_2$ being split. It is well known that splitting preserves satisfiability of tableaux. We may choose to apply splitting eagerly, or lazily or in some other fashion. Hence we define a *splitting strategy* to be a function f such that $\mathcal{T} \rightarrow_{spl} f(\mathcal{T})$ for all tableaux $\mathcal{T}$. The relation $\Rightarrow_<$ is extended to tableaux as expected. Ordered resolution with splitting strategy is then defined by the following rule: $\mathcal{T}_1 \Rightarrow_{<,f} f(\mathcal{T}_2)$ if $\mathcal{T}_1 \Rightarrow_< \mathcal{T}_2$. This provides us with a well known sound and complete method for testing satisfiability. For any binary relation R, R^* will denote the reflexive transitive closure of R, and R^+ will denote the transitive closure of R.

Lemma 1 ([2]). *For any set S of clauses, for any stable ordering $<$, and for any splitting strategy f, S is unsatisfiable iff $S \Rightarrow^*_{<,f} \mathcal{T}$ for some closed $\mathcal{T}$.*

If all predicates are zero-ary then the resulting clauses are *propositional clauses*. In this case we write $S \models_{\mathrm{p}} T$ to say that every Herbrand model of S is a Herbrand model of T. This notation will also be used when S and T are sets of first order clauses, by treating every (ground or non-ground) atom as a zero-ary predicate. For example $\{P(a), -P(a)\} \models_{\mathrm{p}} \square$ but $\{P(x), -P(a)\} \not\models_{\mathrm{p}} \square$. $S \models_{\mathrm{p}} \{C\}$ is also written as $S \models_{\mathrm{p}} C$. If $S \models_{\mathrm{p}} C$ then clearly $S\sigma \models_{\mathrm{p}} C\sigma$ for all substitution σ.

3 Cryptographic Protocols

We assume that Σ contains the binary functions $\{\_\}\_$ and $\langle \_, \_ \rangle$ denoting encryption and pairing. *Messages* are terms of $T_\Sigma(\mathbf{X})$. A *state* is of the form $S(M_1, \ldots, M_n)$ where S with arity n is from a finite set of *control points* and M_i are messages. It denotes an agent at control point S with messages M_i in its memory. An *initialization state* is a state not containing variables. A *protocol rule* is of the form $S_1(M_1, \ldots, M_m) : \mathsf{recv}(M) \rightarrow S_2(N_1, \ldots, N_n) : \mathsf{send}(N)$. Here M_i, N_j are messages, and M and N are each either a message, or a dummy symbol $?$ indicating nothing is received (resp. sent). For secrecy analysis we can replace $?$ by some public message, i.e. one which is known to everyone including the adversary. The rule says that an agent in state $S_1(M_1, \ldots, M_m)$ can receive message M, send a message N, and then move to state $S_2(N_1, \ldots, N_n)$, thus also modifying the messages in its memory. A *protocol* is a finite set of initialization states and protocol rules. This model is in the style of [9] and [5]. The assumption of single blind copying then says that each protocol rule contains at most one variable (which may occur anywhere any number of times in that rule). For example, the public-key Needham-Schroeder protocol below

$$\begin{aligned} A &\rightarrow B : \{A, N_A\}_{K_B} \\ B &\rightarrow A : \{N_A, N_B\}_{K_B} \\ A &\rightarrow B : \{N_B\}_{K_B} \end{aligned}$$

is written in our notation as follows. For every pair of agents A and B in our system (finitely many of them suffice for finding all attacks against secrecy [7, 6]) we have two nonces N^1_{AB} and N^2_{AB} to be used in sessions where A plays the initiator's role and B plays the responder's role. We have initialization states $\mathsf{Init}_0(A, N^1_{AB})$ and $\mathsf{Resp}_0(B, N^2_{AB})$ for all agents A and B. Corresponding to the three lines in the protocol we have rules for all agents A and B:

$$\begin{aligned} \mathsf{Init}_0(A, N^1_{AB}){:}\mathsf{recv}(?) &\rightarrow \mathsf{Init}_1(A, N^1_{AB}){:}\mathsf{send}(\{\langle A, N^1_{AB}\rangle\}_{K_B}) \\ \mathsf{Resp}_0(B, N^2_{AB}){:}\mathsf{recv}(\{\langle A, x\rangle\}_{K_B}) &\rightarrow \mathsf{Resp}_1(B, x, N^2_{AB}){:}\mathsf{send}(\{\langle x, N^2_{AB}\rangle\}_{K_A}) \\ \mathsf{Init}_1(A, N^1_{AB}){:}\mathsf{recv}(\{\langle N^1_{AB}, x\rangle\}_{K_A}) &\rightarrow \mathsf{Init}_2(A, N^1_{AB}, x){:}\mathsf{send}(\{x\}_{K_B}) \\ \mathsf{Resp}_1(B, x, N^2_{AB}){:}\mathsf{recv}(\{N^2_{AB}\}_{K_B}) &\rightarrow \mathsf{Resp}_2(B, x, N^2_{AB}){:}\mathsf{send}(?) \end{aligned}$$

Any initialization state can be created any number of times and any protocol rule can be executed any number of times. The adversary has full control over the network: all messages received by agents are actually sent by the adversary and all messages sent by agents are actually received by the adversary. The adversary can obtain new messages from messages he knows, e.g. by performing encryption and decryption. To model this using Horn clauses, we create a unary predicate reach to model reachable

states, and a unary predicate known to model messages known to the adversary. The initialization state $S(M_1, \ldots, M_n)$ is then modeled by the clause $\mathsf{reach}(S(M_1, \ldots, M_n))$, where S is a new function symbol we create. The protocol rule $S_1(M_1, \ldots, M_m) : \mathsf{recv}(M) \rightarrow S_2(N_1, \ldots, N_n) : \mathsf{send}(N)$ is modeled by the clauses $\mathsf{known}(N) \vee -\mathsf{reach}(S_1(M_1, \ldots, M_m)) \vee -\mathsf{known}(M)$ and $\mathsf{reach}(S_2(N_1, \ldots, N_n)) \vee -\mathsf{reach}(S_1(M_1, \ldots, M_m)) \vee -\mathsf{known}(M)$. Under the assumption of single blind copying it is clear that all these clauses are *one-variable clauses*, i.e. clauses containing at most one variable. We need further clauses to express adversary capabilities. The clauses $\mathsf{known}(\{\mathbf{x}_1\}_{\mathbf{x}_2}) \vee -\mathsf{known}(\mathbf{x}_1) \vee -\mathsf{known}(\mathbf{x}_2)$ and $\mathsf{known}(\mathbf{x}_1) \vee -\mathsf{known}(\{\mathbf{x}_1\}_{\mathbf{x}_2}) \vee -\mathsf{known}(\mathbf{x}_2)$ express the encryption and decryption abilities of the adversary. We have similar clauses for his pairing and unpairing abilities, as well as clauses $\mathsf{known}(f(\mathbf{x}_1, \ldots, \mathbf{x}_n)) \vee -\mathsf{known}(\mathbf{x}_1) \vee \ldots \vee -\mathsf{known}(\mathbf{x}_n)$ for any function f that the adversary knows to apply. All these are clearly *flat clauses*, i.e. clauses of the form $C = \bigvee_{i=1}^{k} \pm_i P_i(f_i(x_1^i, \ldots, x_{n_i}^i)) \vee \bigvee_{j=1}^{l} \pm_j Q_j(x_j)$, where $\{x_1^i, \ldots, x_{n_i}^i\} = \mathsf{fv}(C)$ for $1 \leq i \leq k$. Asymmetric keys, i.e. keys K such that message $\{M\}_K$ can only be decrypted with the inverse key K^{-1}, are also easily dealt with using flat and one-variable clauses. The adversary's knowledge of other data c like agent's names, public keys, etc are expressed by clauses $\mathsf{known}(c)$. Then the least Herbrand model of this set of clauses describes exactly the reachable states and the messages known to the adversary. Then to check whether some message M remains secret, we add the clause $-\mathsf{known}(M)$ and check whether the resulting set is satisfiable.

A set of clauses is in the class $\mathcal{V}_1$ if each of its members is a one-variable clause. A set of clauses is in the class $\mathcal{F}$ if each of its members is a flat clause. More generally we have the class $\mathcal{C}$ proposed by Comon and Cortier [6, 8]: a set of clauses S is in the class $\mathcal{C}$ if for each $C \in S$ one of the following conditions is satisfied:
– C is a one-variable clause
– $C = \bigvee_{i=1}^{k} \pm_i P_i(u_i[f_i(x_1^i, \ldots, x_{n_i}^i)]) \vee \bigvee_{j=1}^{l} \pm_j Q_j(x_j)$, where for $1 \leq i \leq k$ we have $\{x_1^i, \ldots, x_{n_i}^i\} = \mathsf{fv}(C)$ and u_i contains at most one variable.
If all clauses are Horn then we have the corresponding classes $\mathcal{V}_1 Horn$, $\mathcal{F} Horn$ and $\mathcal{C} Horn$. Clearly the classes $\mathcal{V}_1$ (resp. $\mathcal{V}_1 Horn$) and $\mathcal{F}$ (resp. $\mathcal{F} Horn$) are included in the class $\mathcal{C}$ (resp. $\mathcal{C} Horn$) since the u_i's above can be trivial. Conversely any clause set in $\mathcal{C}$ can be considered as containing just flat and one-variable clauses. This is because we can replace a clause $C \vee \pm P(u[f(x_1, \ldots, x_n)])$ by the clause $C \vee \pm Pu(f(x_1, \ldots, x_n))$ and add clauses $-Pu(x) \vee P(u[x])$ and $Pu(x) \vee -P(u[x])$ where Pu is a fresh predicate. This transformation takes polynomial time and preserves satisfiability of the clause set. Hence now we need to deal with just flat and one-variable clauses. In the rest of the paper we derive optimal complexity results for all these classes.

Still this only gives us an upper bound for the secrecy problem of protocols since the clauses could be more general than necessary. It turns out, however, that this is not the case. In order to show this we rely on a reduction of the reachability problem for *alternating pushdown systems (APDS)*. In form of Horn clauses, an *APDS* is a finite set of clauses of the form (i) $P(a)$ where a is a zero-ary symbol, (ii) $P(s[x]) \vee -Q(t[x])$ where s and t involve only unary function symbols, and (iii) $P(x) \vee -P_1(x) \vee -P_2(x)$. Given such an APDS S, a ground atom $P(t)$ is *reachable* if $P(t)$ is in the least Herbrand model of S, i.e. if $S \cup \{-P(t)\}$ is unsatisfiable. Reachability in APDS is DEXPTIME-

hard [4]. We encode this problem into secrecy of protocols, as in [9]. Let K be a (symmetric) key not known to the adversary. Encode atoms $P(t)$ as messages $\{\langle P, t\rangle\}_K$, by treating P as some data. Create an initialization state S (no message is stored in the state). Clause (i) is translated as $S : \mathsf{recv}(?) \rightarrow S : \mathsf{send}(\{\langle P, a\rangle\}_K)$. Clause (ii) is translated as $S : \mathsf{recv}(\{\langle Q, t[x]\rangle\}_K) \rightarrow S : \mathsf{send}(\{\langle P, s[x]\rangle\}_K)$. Clause (iii) is translated as $S : \mathsf{recv}(\langle\{\langle P_1, x\rangle\}_K, \{\langle P_2, x\rangle\}_K\rangle) \rightarrow S : \mathsf{send}(\{\langle P, x\rangle\}_K)$. The intuition is that the adversary cannot decrypt messages encrypted with K. He also cannot encrypt messages with K. He can only forward messages which are encrypted with K. However he has the ability to pair messages. This is utilized in the translation of clause (iii). Then a message $\{M\}_K$ is known to the adversary iff M is of the form $\langle P, t\rangle$ and $P(t)$ is reachable in the APDS.

Theorem 1. *Secrecy problem for cryptographic protocols with single blind copying, with bounded number of nonces but unbounded number of sessions is DEXPTIME-hard, even if no message is allowed to be stored at any control point.*

4 One Variable Clauses: Decomposition of Terms

We first show that satisfiability for the classes $\mathcal{V}_1$ and $\mathcal{V}_1 Horn$ is DEXPTIME-complete. Note that although we consider only unary predicates, this is no restriction in the case of one-variable clauses, since we can encode atoms $P(t_1, \ldots, t_n)$ as $P'(f_n(t_1 \ldots, t_n))$ for fresh P' and f_n for every P of arity n. As shown in [6, 8], ordered resolution on one-variable clauses, for a suitable ordering, leads to a linear bound on the height of terms produced. This does not suffice for obtaining a DEXPTIME upper bound and we need to examine the forms of unifiers produced during resolution. We consider terms containing at most one variable (call them *one-variable terms*) to be compositions of simpler terms. A non-ground one-variable term $t[x]$ is called *reduced* if it is not of the form $u[v[x]]$ for any non-ground non-trivial one-variable terms $u[x]$ and $v[x]$. The term $f(g(x), h(g(x)))$ for example is not reduced because it can be written as $f(x, h(x))[g(x)]$. The term $f'(x, g(x), a)$ is reduced. Unifying it with the reduced term $f'(h(y), g(h(a)), y)$ produces ground unifier $\{x \mapsto h(y)[a], y \mapsto a\}$ and both $h(y)$ and a are strict subterms of the given terms. Indeed we find:

Lemma 2. *Let $s[x]$ and $t[y]$ be reduced, non-ground and non-trivial terms where $x \neq y$ and $s[x] \neq t[x]$. If s and t have a unifier σ then $x\sigma, y\sigma \in U[V]$ where U is the set of non-ground (possibly trivial) strict subterms of s and t, and V is the set of ground strict subterms of s and t.*

In case both terms (even if not reduced) have the same variable we have the following easy result:

Lemma 3. *Let σ be a unifier of two non-trivial, non-ground and distinct one-variable terms $s[x]$ and $t[x]$. Then $x\sigma$ is a ground strict subterm of s or of t.*

In the following one-variable clauses are simplified to involve only reduced terms.

Lemma 4. *Any non-ground one-variable term $t[x]$ can be uniquely written as $t[x] = t_1[t_2[\ldots[t_n[x]]\ldots]]$ where $n \geq 0$ and each $t_i[x]$ is non-trivial, non-ground and reduced. This decomposition can be computed in time polynomial in the size of t.*

Above and elsewhere, if $n = 0$ then $t_1[t_2[\ldots[t_n[x]]\ldots]]$ denotes x. Now if a clause set contains a clause $C = C' \vee \pm P(t[x])$, with $t[x]$ being non-ground, if $t[x] = t_1[\ldots[t_n[x]]\ldots]$ where each t_i is non-trivial and reduced, then we create fresh predicates $Pt_1 \ldots t_i$ for $1 \leq i \leq p-1$ and replace C by the clause $C' \vee \pm Pt_1 \ldots t_{n-1}(t_n[x])$. Also we add clauses $Pt_1 \ldots t_i(t_{i+1}[x]) \vee -Pt_1 \ldots t_{i+1}(x)$ and $-Pt_1 \ldots t_i(t_{i+1}[x]) \vee Pt_1 \ldots t_{i+1}(x)$ for $0 \leq i \leq n-2$ to our clause set. Note that the predicates $Pt_1 \ldots t_i$ are considered invariant under renaming of terms t_j. For $i = 0$, $Pt_1 \ldots t_i$ is same as P. Our transformation preserves satisfiability of the clause set. By Lemma 4 this takes polynomial time and eventually all non-ground literals in clauses are of the form $\pm P(t)$ with reduced t. Next if the clause set is of the form $S \cup \{C_1 \cup C_2\}$, where C_1 is non-empty and has only ground literals, and C_2 is non-empty and has only non-ground literals, then we do splitting to produce $S \cup \{C_1\} \mid S \cup \{C_2\}$. This process produces at most exponentially many branches each of which has polynomial size. Now it suffices to decide satisfiability of each branch in DEXPTIME. Hence now we assume that each clause is either:

(**Ca**) a ground clause, or

(**Cb**) a clause containing exactly one variable, each of whose literals is of the form $\pm P(t[x])$ where t is non-ground and reduced.

Consider a set S of clauses of type Ca and Cb. We show how to decide satisfiability of the set S. Wlog we assume that all clauses in S of type Cb contain the variable $\mathbf{x}_1$. Let Ng be the set of non-ground terms $t[\mathbf{x}_1]$ occurring as arguments in literals in S. Let Ngs be the set of non-ground subterms $t[\mathbf{x}_1]$ of terms in Ng. We assume that Ng and Ngs always contain the trivial term $\mathbf{x}_1$, otherwise we add this term to both sets. Let G be the set of ground subterms of terms occurring as arguments in literals in S. The sizes of Ng, Ngs and G are polynomial. Let $S^\dagger$ be the set of clauses of type Ca and Cb which only contain literals of the form $\pm P(t)$ for some $t \in \mathsf{Ng} \cup \mathsf{Ng}[\mathsf{Ngs}[\mathsf{G}]]$ (observe that $\mathsf{G} \subseteq \mathsf{Ngs}[\mathsf{G}] \subseteq \mathsf{Ng}[\mathsf{Ngs}[\mathsf{G}]]$). The size of $S^\dagger$ is at most exponential.

For resolution we use ordering $\prec$: $P(s) \prec Q(t)$ iff s is a strict subterm of t. We call $\prec$ the subterm ordering without causing confusion. This is clearly stable. This is the ordering that we are going to use throughout this paper. In particular this means that if a clause contains literals $\pm P(x)$ and $\pm' Q(t)$ where t is non-trivial and contains x, then we cannot choose the literal $\pm P(x)$ to resolve upon in this clause. Because of the simple form of unifiers of reduced terms we have:

Lemma 5. *Binary ordered resolution and ordered factorization, wrt the subterm ordering, on clauses in $S^\dagger$ produces clauses which are again in $S^\dagger$ (upto renaming).*

Hence to decide satisfiability of $S \subseteq S^\dagger$, we keep generating new clauses of $S^\dagger$ by doing ordered binary resolution and ordered factorization wrt the subterm ordering till no new clause can be generated, and then check whether the empty clause has been produced. Also recall that APDS consist of Horn one-variable clauses. Hence:

Theorem 2. *Satisfiability for the classes $\mathcal{V}_1$ and $\mathcal{V}_1 Horn$ is DEXPTIME-complete.*

5 Flat Clauses: Resolution Modulo Propositional Reasoning

Next we show how to decide the class $\mathcal{F}$ of flat clauses in NEXPTIME. This is well known when the maximal arity r is a constant, or when all non-trivial literals in a

clause have the same *sequence* (instead of the same *set*) of variables. But we are not aware of a proof of NEXPTIME upper bound in the general case. We show how to obtain NEXPTIME upper bound in the general case, by doing resolution modulo propositional reasoning. While this constitutes an interesting result of its own, the techniques allow us to deal with the full class $\mathcal{C}$ efficiently. Also this shows that the generality of the class $\mathcal{C}$ does not cost more in terms of complexity. An ϵ-*block* is a one-variable clause which contains only trivial literals. A complex clause C is a flat clause $\bigvee_{i=1}^{k} \pm_i P_i(f_i(x_1^i, \ldots, x_{n_i}^i)) \vee \bigvee_{j=1}^{l} \pm_j Q_j(x_j)$ in which $k \geq 1$. A flat clause is either a complex clause, or an ϵ-*clause* which is defined to be a disjunction of ϵ-blocks, i.e. to be of the form $C_1[x_1] \sqcup \ldots \sqcup C_n[x_n]$ where each C_i is an ϵ-block. ϵ-clauses are difficult to deal with, hence we split them to produce ϵ-blocks. Hence define ϵ-*splitting* as the restriction of the splitting rule in which one of the components is an ϵ-block.

Recall that r is the maximal arity of symbols in Σ. Any complex clause C can be renamed to make it *good* i.e. such that $\mathsf{fv}(C) \subseteq \mathbf{X}_r = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\}$. An ϵ-block C can be renamed to make it *good* i.e. of the form $C[\mathbf{x}_{r+1}]$. The choice of $\mathbf{x}_{r+1}$ is not crucial. Now notice that ordered resolution between complex clauses and ϵ-blocks only produces flat clauses, which can then be split to be left with only complex and ϵ-blocks. E.g. Resolution between $P_1(\mathbf{x}_1) \vee -P_2(\mathbf{x}_2) \vee P_3(f(\mathbf{x}_1, \mathbf{x}_2)) \vee -P_4(g(\mathbf{x}_2, \mathbf{x}_1))$ and $P_4(g(\mathbf{x}_1, \mathbf{x}_1)) \vee -P_5(h(\mathbf{x}_1)) \vee P_6(\mathbf{x}_1)$ produces $P_1(\mathbf{x}_1) \vee -P_2(\mathbf{x}_1) \vee P_3(f(\mathbf{x}_1, \mathbf{x}_1)) \vee -P_5(h(\mathbf{x}_1)) \vee P_6(\mathbf{x}_1)$. Resolution between $P_2(\mathbf{x}_{r+1})$ and $-P_2(f(\mathbf{x}_1, \mathbf{x}_2)) \vee P_3(\mathbf{x}_1) \vee P_4(\mathbf{x}_2)$ produces $P_3(\mathbf{x}_1) \vee P_4(\mathbf{x}_2)$ which can then be split. The point is that we always choose a non-trivial literal from a clause for resolution, if there is one. As there are finitely many complex clauses and ϵ-blocks this gives us a decision procedure. Note however that the number of complex clauses is doubly exponential. This is because we allow clauses of the form $P_1(f_1(\mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_2)) \vee P_2(f_2(\mathbf{x}_2, \mathbf{x}_1)) \vee P_3(f_3(\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_2)) \vee \ldots$, i.e. the nontrivial terms contain arbitrary number of repetitions of variables in arbitrary order. The number of such variable sequences of r variables is exponentially many, hence the number of clauses is doubly exponential. Letting the maximal arity r to be a constant, or forcing all non-trivial literals in a clause to have the same variable sequence would have produced only exponentially many clauses. In presence of splitting, this would have given us the well-known NEXPTIME upper bound, which is also optimal. But we are not aware of a proof of NEXPTIME upper bound in the general case. To obtain NEXPTIME upper bound in the general case we introduce the technique of resolution modulo propositional reasoning.

For a clause C, define the set of its projections as $\pi(C) = C[\mathbf{X}_r]$. Essentially projection involves making certain variables in a clause equal. As we saw, resolution between two complex clauses amounts to propositional resolution between their projections. Define the set $\mathsf{U} = \{f(x_1, \ldots, x_n) \mid f \in \Sigma \text{ and each } x_i \in \mathbf{X}_r\}$ of size exponential in r. Resolution between ϵ-block C_1 and a good complex clause C_2 amounts to propositional resolution of a clause from $C[\mathsf{U}]$ with C_2. Also note that propositional resolution followed by further projection is equivalent to projection followed by propositional resolution. Each complex clause has exponentially many projections. This suggests that we can compute beforehand the exponentially many projections of complex clauses and exponentially many instantiations of ϵ-blocks. All new complex clauses generated by propositional resolution are ignored. But after several such propositional

resolution steps, we may get an ϵ-clause, which should then be split and instantiated and used for obtaining further propositional resolvents. In other words we only compute such propositionally implied ϵ-clauses, do splitting and instantiation and iterate the process. This generates all resolvents upto propositional implication. The difference from the approach of Ganzinger and Korovin [10] is that they have a single phase of instantiation followed by propositional satisfiability checking. In contrast, we compute certain interesting propositional implications which are further instantiated, and iterate the process. We now formalize our approach.

For a set S of clauses, let $\mathsf{comp}(S)$ be the set of complex clauses in S, $\mathsf{eps}(S)$ be the set of ϵ-blocks in S, $\pi(S) = \bigcup_{C \in S} \pi(C)$ and $\mathsf{I}(S) = S \cup \pi(\mathsf{comp}(S)) \cup \mathsf{eps}(S)[\mathsf{U}]$. For sets S and T of complex clauses and ϵ-blocks, write $S \sqsubseteq T$ to mean that:
– if C is a complex clause in S then $\mathsf{I}(T) \vDash_{\mathrm{p}} \pi(C)$, and
– every ϵ-block in S can be renamed as some $C[\mathbf{x}_{r+1}] \in T$.
For tableaux $\mathcal{T}_1$ and $\mathcal{T}_2$ involving only complex clauses and ϵ-blocks we write $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ if $\mathcal{T}_1$ can be written as $S_1 \mid \ldots \mid S_n$ and $\mathcal{T}_2$ can be written as $T_1 \mid \ldots \mid T_n$ (note same n) such that $S_i \sqsubseteq T_i$ for $1 \leq i \leq n$. Intuitively $\mathcal{T}_2$ is a succinct representation of $\mathcal{T}_1$. Define the splitting strategy f as the one which repeatedly applies ϵ-splitting on a tableau as long as possible. The relation $\Rightarrow_{\prec,f}$ provides us a sound and complete method for testing unsatisfiability. We define the alternative procedure for testing unsatisfiability by using succinct representations of tableaux. We define $\blacktriangleright$ by the rule: $\mathcal{T} \mid S \blacktriangleright \mathcal{T} \mid S \cup \{C_1[\mathbf{x}_{r+1}]\} \mid \ldots \mid S \cup \{C_k[\mathbf{x}_{r+1}]\}$ whenever $\mathsf{I}(S) \vDash_{\mathrm{p}} C = C_1[\mathbf{x}_{i_1}] \sqcup \ldots \sqcup C_k[\mathbf{x}_{i_k}]$, C is an ϵ-clause, and $1 \leq i_1, \ldots, i_k \leq r$. Then $\blacktriangleright$ simulates $\Rightarrow_{\prec,f}$:

Lemma 6. *If S is a set of complex clauses and ϵ-blocks, $S \sqsubseteq T$ and $S \Rightarrow_{\prec,f} \mathcal{T}$, then all clauses occurring in $\mathcal{T}$ are complex clauses or ϵ-blocks and $T \blacktriangleright^* \mathcal{T}'$ for some $\mathcal{T}'$ such that $\mathcal{T} \sqsubseteq \mathcal{T}'$.*

Hence we have completeness of $\blacktriangleright$:

Lemma 7. *If a set S of good complex clauses and ϵ-blocks is unsatisfiable then $S \blacktriangleright^* \mathcal{T}$ for some closed $\mathcal{T}$.*

Proof. By Lemma 1, $S \Rightarrow^*_{\prec,f} S_1 \mid \ldots \mid S_n$ such that each $S_i \ni \Box$. Since all complex clauses and ϵ-blocks in S are good, we have $S \sqsubseteq S$. Hence by Lemma 6, we have some $T_1, \ldots, T_n$ such that $S \blacktriangleright^* T_1 \mid \ldots \mid T_n$ and $S_i \sqsubseteq T_i$ for $1 \leq i \leq n$. Since $\Box \in S_i$ and $\Box$ is an ϵ-block, hence $\Box \in T_i$ for $1 \leq i \leq n$. $\Box$

Call a set S of good complex clauses and ϵ-blocks *saturated* if the following condition is satisfied: if $\mathsf{I}(S) \vDash_{\mathrm{p}} B_1[\mathbf{x}_{i_1}] \sqcup \ldots \sqcup B_k[\mathbf{x}_{i_k}]$ with $1 \leq i_1, \ldots, i_k \leq r$, each B_i being an ϵ-block, then there is some $1 \leq j \leq k$ such that $B_j[\mathbf{x}_{r+1}] \in S$.

Lemma 8. *If S is a satisfiable set of good complex clauses and ϵ-blocks then $S \blacktriangleright^* \mathcal{T} \mid T$ for some $\mathcal{T}$ and some saturated set T of good complex clauses and ϵ-blocks, such that $\Box \notin T$.*

Proof. We construct a sequence $S = S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$ of good complex clauses and ϵ-blocks such that S_i is satisfiable and $S_i \blacktriangleright^* S_{i+1} \mid \mathcal{T}_i$ for some $\mathcal{T}_i$ for each i. $S = S_0$ is satisfiable by assumption. Now assume we have already defined $S_0, \ldots, S_i$

and $\mathcal{T}_0, \ldots, \mathcal{T}_{i-1}$. Let $C^l = B^l_1[\mathbf{x}_{i^l_1}] \sqcup \ldots \sqcup B^l_k[\mathbf{x}_{i^l_{k_l}}]$ for $1 \leq l \leq N$ be all the possible ϵ-clauses such that $\mathsf{I}(S_i) \models_{\mathrm{p}} C^l$, $1 \leq i^l_1, \ldots, i^l_{k_l} \leq r$. Since S_i is satisfiable, $S_i \cup \{C^l \mid 1 \leq l \leq N\}$ is satisfiable. Since $\mathbf{x}_{i^l_1}, \ldots, \mathbf{x}_{i^l_{k_l}}$ are mutually distinct for $1 \leq l \leq N$, there are $1 \leq j_l \leq k_l$ for $1 \leq l \leq N$ such that $S_i \cup \{B^l_{j_l}[\mathbf{x}_{i^l_{j_l}}] \mid 1 \leq l \leq N\}$ is satisfiable. Let $S_{i+1} = S_i \cup \{B^l_{j_l}[\mathbf{x}_{r+1}] \mid 1 \leq l \leq N\}$. S_{i+1} is satisfiable. Also it is clear that $S_i \blacktriangleright^* S_{i+1} \mid \mathcal{T}_i$ for some $\mathcal{T}_i$. If $S_{i+1} = S_i$ then S_i is saturated, otherwise S_{i+1} has strictly more ϵ-blocks. As there are only finitely many good ϵ-blocks, eventually we will end up with a saturated set T in this way. Since T is satisfiable, $\square \notin T$. From construction it is clear that there is some $\mathcal{T}$ such that $S \blacktriangleright^* \mathcal{T} \mid T$. $\square$

Theorem 3. *Satisfiability for the class $\mathcal{F}$ is NEXPTIME-complete.*

Proof. The lower bound comes from reduction of satisfiability of positive set constraints which is NEXPTIME-complete [1]. For the upper bound let S be a finite set of flat clauses. Repeatedly apply ϵ-splitting to obtain $f(S) = S_1 \mid \ldots \mid S_m$. S is satisfiable iff some S_i is satisfiable. The number m of branches in $f(S)$ is at most exponential. Also each branch has size linear in the size of S. We non-deterministically choose some S_i and check its satisfiability in NEXPTIME.

Hence wlog we may assume that the given set S has only complex clauses and ϵ-blocks. Wlog all clauses in S are good. We non-deterministically choose a certain number of good ϵ-blocks $B_1[\mathbf{x}_{r+1}], \ldots, B_N[\mathbf{x}_{r+1}]$ and check that $T = S_1 \cup \{B_1[\mathbf{x}_{r+1}], \ldots, B_N[\mathbf{x}_{r+1}]\}$ is saturated and $\square \notin T$. By Lemma 8, if S is satisfiable then clearly there is such a set T. Conversely if there is such a set T, then whenever $T \blacktriangleright^* \mathcal{T}$, we will have $\mathcal{T} = T \mid \mathcal{T}'$ for some $\mathcal{T}'$. Hence we can never have $T \blacktriangleright^* \mathcal{T}$ where $\mathcal{T}$ is closed. Then by Lemma 7 we conclude that T is satisfiable. Hence $S \subseteq T$ is also satisfiable.

Guessing the set T requires non-deterministically choosing from among exponentially many ϵ-blocks. To check that T is saturated, for every ϵ-clause $C = B_1[\mathbf{x}_{i_1}] \sqcup \ldots \sqcup B_k[\mathbf{x}_{i_k}]$, with $1 \leq i_1, \ldots, i_k \leq r$, and $B_j[\mathbf{x}_{r+1}] \notin T$ for $1 \leq j \leq k$, we check that $\mathsf{I}(T) \not\models_{\mathrm{p}} C$, i.e. $\mathsf{I}(T) \cup \neg C$ is propositionally satisfiable (where $\neg(L_1 \vee \ldots \vee L_n)$ denotes $\{-L_1, \ldots, -L_n\}$). This can be checked in NEXPTIME since propositional satisfiability can be checked in NPTIME. We need to do such checks for at most exponentially many possible values of C. $\square$

6 Combination: Ordered Literal Replacement

Combining flat and one-variable clauses creates additional difficulties. First observe that resolving a one variable clause $C_1 \vee \pm P(f(s_1[x], \ldots, s_n[x]))$ with a complex clause $\mp P(f(x_1, \ldots, x_n)) \vee C_2$ produces a one-variable clause. If $s_i[x] = s_j[x]$ for all $x_i = x_j$, and if C_2 contains a literal $P(x_i)$ then the resolvent contains a literal $P(s_i[x])$. The problem now is that even if $f(s_1[x], \ldots, s_n[x])$ is reduced, $s_i[x]$ may not be reduced. E.g. $f(g(h(x)), x)$ is reduced but $g(h(x))$ is not reduced. Like in Section 4 we may think of replacing this literal by simpler literals involving fresh predicates. Firstly we have to ensure that in this process we would not generate infinitely many predicates. Secondly it is not clear that mixing ordered resolution steps with replacement of literals is still complete. Correctness is easy to show since the new clause is in some sense

equivalent to the old deleted clause. However deletion of clauses arbitrarily can violate completeness of the resolution procedure. The key factor which preserves completeness is that we replace literals by smaller literals wrt the given ordering $<$.

Formally a *replacement rule* is of the form $A_1 \rightarrow A_2$ where A_1 and A_2 are (not necessarily ground) atoms. The clause set *associated* with this rule is $\{A_1 \vee -A_2, -A_1 \vee A_2\}$. Intuitively such a replacement rule says that A_1 and A_2 are equivalent. The clause set $cl(\mathcal{R})$ associated with a set $\mathcal{R}$ of replacement rules is the union of the clause sets associated with the individual replacement rules in $\mathcal{R}$. Given a stable ordering $<$ on atoms, a replacement rule $A_1 \rightarrow A_2$ is *ordered* iff $A_2 < A_1$. We define the relation $\rightarrow_{\mathcal{R}}$ as: $S \rightarrow_{\mathcal{R}} (S \setminus \{\pm A_1\sigma \vee C\}) \cup \{\pm A_2\sigma \vee C\}$ whenever S is a set of clauses, $\pm A_1\sigma \vee C \in S$, $A_1 \rightarrow A_2 \in \mathcal{R}$ and σ is some substitution. Hence we replace literals in a clause by smaller literals. The relation is extended to tableaux as usual. This is reminiscent of the well-studied case of resolution with some equational theory on terms. There, however, the ordering $<$ used for resolution is compatible with the equational theory and one essentially works with the equivalence classes of terms and atoms. This is not the case here.

Next note that in the above resolution example, even if $f(s_1[x], \ldots, s_n[x])$ is non-ground, some s_i may be ground. Hence the resolvent may have ground as well as non-ground literals. We avoided this in Section 4 by initial preprocessing. Now we may think of splitting these resolvents during the resolution procedure. This however will be difficult to simulate using the alternative resolution procedure on succinct representations of tableaux because we will generate doubly exponentially many one-variable clauses. To avoid this we use a variant of splitting called *splitting-with-naming* [13]. Instead of creating two branches after splitting, this rule puts both components into the same set, but with tags to simulate branches produced by ordinary splitting. Fix a finite set $\mathbb{P}$ of predicate symbols. $\mathbb{P}$-clauses are clauses whose predicates are all from $\mathbb{P}$. Introduce fresh zero-ary predicates $\overline{C}$ for $\mathbb{P}$-clauses C modulo renaming, i.e. $\overline{C_1} = \overline{C_2}$ iff $C_1\sigma = C_2$ for some renaming σ. Literals $\pm\overline{C}$ for $\mathbb{P}$-clauses C are *splitting literals*. The *splitting-with-naming* rule is defined as: $S \rightarrow_{nspl} (S \setminus \{C_1 \sqcup C_2\}) \cup \{C_1 \vee -\overline{C_2}, \overline{C_2} \vee C_2\}$ where $C_1 \sqcup C_2 \in S$, C_2 is non-empty and has only non-splitting literals, and C_1 has at least one non-splitting literal. Intuitively $\overline{C_2}$ represents the negation of C_2. We will use both splitting and splitting-with-naming according to some predefined strategy. Hence for a finite set $\mathcal{Q}$ of splitting atoms, define *$\mathcal{Q}$-splitting* as the restriction of the splitting-with-naming rule where the splitting atom produced is restricted to be from $\mathcal{Q}$. Call this restricted relation as $\rightarrow_{\mathcal{Q}-nspl}$. This is extended to tableaux as usual. Now once we have generated the clauses $C_1 \vee -\overline{C_2}$ and $\overline{C_2} \vee C_2$ we would like to keep resolving on the second part of the second clause till we are left with the clause $\overline{C_2}$ (possibly with other positive splitting literals) which would then be resolved with the first clause to produce C_1 (possibly with other positive splitting literals) and only then the literals in C_1 would be resolved upon. Such a strategy cannot be ensured by ordered resolution, hence we introduce a new rule. An ordering $<$ over non-splitting atoms is extended to the ordering $<_s$ by letting $q <_s A$ whenever q is a splitting atom and A is a non-splitting atom, and $A <_s B$ whenever A, B are non-splitting atoms and $A < B$. We

define *modified ordered binary resolution* by the following rule:

$$\frac{C_1 \vee A \quad \neg B \vee C_2}{C_1\sigma \vee C_2\sigma}$$

where $\sigma = mgu(A, B)$ and the following conditions are satisfied:
(1) C_1 has no negative splitting literal, and A is maximal in C_1.
(2) (a) either $B \in \mathcal{Q}$, or
(b) C_2 has no negative splitting literal, and B is maximal in C_2.
As usual we rename the premises before resolution so that they don't share variables. This rule says that we must select a negative splitting literal to resolve upon in any clause, provided the clause has at least one such literal. If no such literal is present in the clause, then the ordering $<_s$ enforces that a positive splitting literal will not be selected as long as the clause has some non-splitting literal. We write $S \Rrightarrow_{<_s} S \cup \{C\}$ to say that C is obtained by one application of the modified binary ordered resolution or the (unmodified) ordered factorization rule on clauses in S. This is extended to tableaux as usual. A *$\mathcal{Q}$-splitting-replacement strategy* is a function f such that $\mathcal{T}(\rightarrow_{\mathcal{Q}-nspl} \cup \rightarrow_{spl} \cup \rightarrow_{\mathcal{R}})^* f(\mathcal{T})$ for any tableaux $\mathcal{T}$. Hence we allow both normal splitting and $\mathcal{Q}$-splitting. Modified ordered resolution with $\mathcal{Q}$-splitting-replacement strategy f is defined by the relation: $S \Rrightarrow_{<_s,f,\mathcal{R}} f(T)$ whenever $S \Rrightarrow_{<_s} T$. This is extended to tableaux as usual. The above modified ordered binary resolution rule can be considered as an instance of *ordered resolution with selection* [2], which is known to be sound and complete even with splitting and its variants. Our manner of extending $<$ to $<_s$ is essential for completeness. We now show that soundness and completeness hold even under arbitrary ordered replacement strategies. It is not clear if such rules have been studied elsewhere. Wlog we forbid the useless case of replacement rules containing splitting symbols. The relation $<$ is *enumerable* if the set of all ground atoms can be enumerated as $A_1, A_2, \ldots$ such that if $A_i < A_j$ then $i < j$. The subterm ordering is enumerable.

Theorem 4. *Modified ordered resolution, wrt a stable and enumerable ordering, with $\mathcal{Q}$-splitting and ordered literal replacement is sound and complete for any strategy. I.e. for any set S of $\mathbb{P}$-clauses, for any strict stable and enumerable partial order $<$ on atoms, for any set $\mathcal{R}$ of ordered replacement rules, for any finite set $\mathcal{Q}$ of splitting atoms, and for any $\mathcal{Q}$-splitting-replacement strategy f, $S \cup cl(\mathcal{R})$ is unsatisfiable iff $S \Rrightarrow^*_{<_s,f,\mathcal{R}} \mathcal{T}$ for some closed $\mathcal{T}$.*

For the rest of this section fix a set $\mathbb{S}$ of one-variable $\mathbb{P}$-clauses and complex $\mathbb{P}$-clauses whose satisfiability we need to decide. Let Ng be the set of non-ground terms occurring as arguments in literals in the one-variable clauses of $\mathbb{S}$. We rename all terms in Ng to contain only the variable $\mathbf{x}_{r+1}$. Wlog assume $\mathbf{x}_{r+1} \in \mathsf{Ng}$. Let Ngs be the set of non-ground subterms of terms in Ng, and $\mathsf{Ngr} = \{s[\mathbf{x}_{r+1}] \mid s \text{ is non-ground and reduced, and for some } t, s[t] \in \mathsf{Ngs}\}$. Define $\mathsf{Ngrr} = \{s_1[\ldots[s_m]\ldots] \mid s_1[\ldots[s_n]\ldots] \in \mathsf{Ngs}, m \leq n, \text{ and each } s_i \text{ is non-trivial and reduced}\}$. Define the set of predicates $\mathbb{Q} = \{Ps \mid P \in \mathbb{P}, s \in \mathsf{Ngrr}\}$. Note that $\mathbb{P} \subseteq \mathbb{Q}$. Define the set of replacement rules $\mathcal{R} = \{Ps_1 \ldots s_{m-1}(s_m[\mathbf{x}_{r+1}]) \rightarrow Ps_1 \ldots s_m([\mathbf{x}_{r+1}]) \mid Ps_1 \ldots s_m \in \mathbb{Q}\}$. They are clearly ordered wrt $\prec$. Let G be the set of ground subterms of terms occurring as arguments in literals in $\mathbb{S}$. For the rest of this section the set of splitting atoms that we are going to use is $\mathcal{Q}_0 = \{\overline{\pm P(t)} \mid P \in \mathbb{P}, t \in \mathsf{G}\}$. Their purpose is to remove

ground literals from a non-ground clause. All sets defined above have polynomial size. We also need the set $\mathsf{Ngr}_1 = \{\mathbf{x}_{r+1}\} \cup \{f(s_1,\ldots,s_n) \mid \exists g(t_1,\ldots,t_m) \in \mathsf{Ngr} \cdot \{s_1,\ldots,s_n\} = \{t_1,\ldots,t_m\}\}$ which has exponential size. These terms are produced by resolution of non-ground one-variable clauses with complex clauses, and are also reduced. In the ground case we have the set $\mathsf{G}_1 = \{f(s_1,\ldots,s_n) \mid \exists g(t_1,\ldots,t_m) \in \mathsf{G} \mid \{s_1,\ldots,s_n\} = \{t_1,\ldots,t_m\}\}$ of exponential size. For a set $\mathbb{P}'$ of predicates and a set U of terms, the set $\mathbb{P}'[U]$ of atoms is defined as usual. For a set V of atoms the set $-V$ and $\pm V$ of literals is defined as usual. The following types of clauses will be required during resolution:

C1 clauses $C \vee D$, where C is an ϵ-block with predicates from $\mathbb{Q}$, and $D \subseteq \pm\mathcal{Q}_0$.
C2 clauses $C \vee D$ where C is a one-variable clause with literals from $\pm\mathbb{Q}(\mathsf{Ngr}_1)$, C has at least one non-trivial literal, and $D \subseteq \pm\mathcal{Q}_0$.
C3 clauses $C \vee D$ where C is a non-empty clause with literals from $\pm\mathbb{Q}(\mathsf{Ngr}_1[\mathsf{Ngrr}[\mathsf{G}_1]])$, and $D \subseteq \pm\mathcal{Q}_0$.
C4 clauses $C \vee D$ where $C = \bigvee_{i=1}^{k} \pm_i P_i(f_i(x_1^i,\ldots,x_{n_i}^i)) \vee \bigvee_{j=1}^{l} \pm_j Q_j(x_j)$ is a complex clause with each $P_i \in \mathbb{Q}$, each $Q_j \in \mathbb{P}$ and $D \subseteq \pm\mathcal{Q}_0$

We have already argued why we need splitting literals in the above clauses, and why we need Ngr_1 instead of Ngr in type C2. In type C3 we have Ngrr in place of the set Ngs that we had in Section 4, to take care of interactions between one-variable clauses and complex clauses. In type C4 the trivial literals involve predicates only from $\mathbb{P}$ (and not $\mathbb{Q}$). This is what ensures that we need only finitely many fresh predicates (those from $\mathbb{Q} \setminus \mathbb{P}$) because these are the literals that are involved in replacements when this clause is resolved with a one-variable clause. The $\mathcal{Q}_0$-splitting steps that we use in this section consist of replacing a tableau $\mathcal{T} \mid S$ by the tableau $\mathcal{T} \mid (S \setminus \{C \vee L\}) \cup \{C \vee -\overline{L}, \overline{L} \vee L\}$, where C is non-ground, $L \in \pm\mathbb{P}(\mathsf{G})$ and $C \vee L \in S$. The replacement steps we are going to use are of the following kind:
(1) replacing clause $C_1[x] = C \vee \pm P(t_1[\ldots[t_n[s[x]]]\ldots])$ by clause $C_2[x] = C \vee \pm P t_1 \ldots t_n(s[x])\}$ where $P \in \mathbb{P}$, $s[\mathbf{x}_{r+1}] \in \mathsf{Ngr}$ is non-trivial, and $t_1[\ldots[t_n]\ldots] \in \mathsf{Ngrr}$. We have $\{C_1[\mathbf{x}_{r+1}]\} \cup cl(\mathcal{R})[\mathsf{Ngrr}] \models_{\mathrm{p}} C_2[\mathbf{x}_{r+1}]$.
(2) replacing ground clause $C_1 = C \vee \pm P(t_1[\ldots[t_n[g]]\ldots])$ by clause $C_2 = C \vee \pm P t_1 \ldots t_n[g]\}$ where $P \in \mathbb{P}, g \in \mathsf{Ngrr}[\mathsf{G}_1]$ and $t_1[\ldots[t_n]\ldots] \in \mathsf{Ngrr}$. This replacement is done only when $t_1[\ldots[t_n[g]]\ldots] \in \mathsf{Ngrr}[\mathsf{Ngrr}[\mathsf{G}_1]] \setminus \mathsf{Ngr}_1[\mathsf{Ngrr}[\mathsf{G}_1]]$. We have $\{C_1\} \cup cl(\mathcal{R})[\mathsf{Ngrr}[\mathsf{Ngrr}[\mathsf{G}_1]]] \models_{\mathrm{p}} C_2$.
Define the $\mathcal{Q}_0$-splitting-replacement strategy f as one which repeatedly applies first ϵ-splitting, then the above $\mathcal{Q}_0$-splitting steps, then the above two replacement steps till no further change is possible. Then $\Rrightarrow_{\prec_s, f, \mathcal{R}}$ gives us a sound and complete method for testing unsatisfiability.

As in Section 5 we now define a succinct representation of tableaux and an alternative resolution procedure for them. As we said, a literal $\overline{L} \in \mathcal{Q}_0$ represents $-L$. Hence for a clause C we define $\underline{C}$ as the clause obtained by replacing every $\pm\overline{L}$ by the literal $\mp L$. This is extended to sets of clauses as usual. As before $\mathsf{U} = \{f(x_1,\ldots,x_n) \mid f \in \Sigma,$ and each $x_i \in \mathbf{X}_r\}$. The functions eps and comp of Section 5 are now extended to return ϵ-blocks and complex clauses respectively, possibly in disjunction with splitting literals. For a set S of clauses, define $\mathsf{ov}(S)$ as the set of clauses of type C2 in S. The function π is as before. We need to define which kinds of instantiations are to be used

to generate propositional implications. For a clause C, define $\mathsf{I}_1(C) = \{C\} \cup C[\mathsf{U}] \cup C[\mathsf{U}[\mathsf{Ngrr} \cup \mathsf{Ngrr}[\mathsf{Ngrr}[\mathsf{G}_1]]]] \cup C[\mathsf{Ngr}_1] \cup C[\mathsf{Ngr}_1[\mathsf{Ngrr}[\mathsf{G}_1]]]$. These are the instantiations necessary for ϵ-blocks. Define $\mathsf{I}_2(C) = \{C\} \cup C[\mathsf{Ngrr}[\mathsf{G}_1]]$. These are necessary for one-variable clauses. Define $\mathsf{I}_3(C) = \{C\}$. Ground clauses require no instantiation. Define $\mathsf{I}_4(C) = \pi(C) \cup C[\mathsf{Ngrr} \cup [\mathsf{Ngrr}[\mathsf{Ngrr}[\mathsf{G}_1]]]]$. These are necessary for complex clauses. For a set S of clauses, define $\mathsf{I}_i(S) = \bigcup_{C \in S} \mathsf{I}_i(C)$. For a set S of clauses of type C1-C4 define $\mathsf{I}(S) = \underline{S} \cup \mathsf{I}_1(\underline{\mathsf{eps}(S)}) \cup \mathsf{I}_2(\underline{\mathsf{ov}(S)}) \cup \mathsf{I}_4(\underline{\mathsf{comp}(S)}) \cup cl(\mathcal{R})[\mathsf{Ngrr} \cup \mathsf{Ngrr}[\mathsf{Ngrr}[\mathsf{G}_1]]]$. Note that instantiations of clauses in $cl(\mathcal{R})$ are necessary for the replacement rules, as argued above. For a set T of clauses define the following properties:
(P1$_T$) C satisfies property P1$_T$ iff $C[\mathbf{x}_{r+1}] \in T$.
(P2$_T$) C satisfies property P2$_T$ iff $\mathsf{I}(T) \models_{\mathrm{p}} \mathsf{I}_2(\underline{C}[\mathbf{x}_{r+1}])$.
(P3$_T$) C satisfies property P3$_T$ iff $\mathsf{I}(T) \models_{\mathrm{p}} \mathsf{I}_3(\underline{C})$.
(P4$_T$) C satisfies property P4$_T$ iff $\mathsf{I}(T) \models_{\mathrm{p}} \mathsf{I}_4(\underline{C})$.
For sets of clauses S and T, define $S \sqsubseteq T$ to mean that every $C \in S$ is of type Ci and satisfies property Pi_T for some $1 \leq i \leq 4$. This is extended to tableaux as usual. The alternative resolution procedure for testing unsatisfiability by using succinct representations of tableaux is now defined by the rule: $\mathcal{T} \mid S \blacktriangleright \mathcal{T} \mid S \cup \{C_1[\mathbf{x}_{r+1}] \sqcup D\} \mid S \cup \{C_2[\mathbf{x}_{r+1}]\} \mid \ldots \mid S \cup \{C_k[\mathbf{x}_{r+1}]\}$ whenever $\mathsf{I}(S) \models_{\mathrm{p}} C_1[\mathbf{x}_{i_1}] \sqcup \ldots \sqcup C_k[\mathbf{x}_{i_k}] \sqcup \underline{D}$, each C_i is an ϵ-block, $1 \leq i_1, \ldots, i_k \leq r$ and $D \subseteq \pm\mathcal{Q}_0$. The simulation property now states:

Lemma 9. *If $S \sqsubseteq T$ and $S \Rightarrow_{\prec_s, f, \mathcal{R}} \mathcal{T}$ then $T \blacktriangleright^* \mathcal{T}'$ for some $\mathcal{T}'$ such that $\mathcal{T} \sqsubseteq \mathcal{T}'$.*

Hence as for flat clauses we obtain:

Theorem 5. *Satisfiability for the class $\mathcal{C}$ is NEXPTIME-complete.*

7 The Horn Case

We show that in the Horn case, the upper bound can be improved to DEXPTIME. The essential idea is that propositional satisfiability of Horn clauses is in PTIME instead of NPTIME. But now we need to eliminate the use of tableaux altogether. To this end, we replace the ϵ-splitting rule of Section 6 by splitting-with-naming. Accordingly we define the set of splitting atoms as $\mathcal{Q} = \mathcal{Q}_0 \cup \mathcal{Q}_1$ where $\mathcal{Q}_1 = \{\overline{C} \mid C$ is a non-empty negative $\epsilon-$block with predicates from $\mathbb{P}\}$. We know that binary resolution and factorization on Horn clauses produces Horn clauses. Replacements on Horn clauses using the rules from $\mathcal{R}$ produces Horn clauses. $\mathcal{Q}_1$-splitting on Horn clauses produces Horn clauses. E.g. clause $P(\mathbf{x}_1) \vee -Q(\mathbf{x}_1) \vee -R(\mathbf{x}_2)$ produces $P(\mathbf{x}_1) \vee -Q(\mathbf{x}_1) \vee --\overline{R(\mathbf{x}_2)}$ and $\overline{-R(\mathbf{x}_2)} \vee -R(\mathbf{x}_2)$. $\mathcal{Q}_0$-splitting on $P(f(x)) \vee -Q(a)$ produces $P(f(\mathbf{x}_1)) \vee --\overline{Q(a)}$ and $\overline{-Q(a)} \vee -Q(a)$ which are Horn. However $\mathcal{Q}_0$-splitting on $C = -P(f(\mathbf{x}_1)) \vee Q(a)$ produces $C_1 = -P(f(\mathbf{x}_1)) \vee -\overline{Q(a)}$ and $C_2 = \overline{Q(a)} \vee Q(a)$. C_2 is not Horn. However $\underline{C_1} = C$ and $\underline{C_2} = -Q(a) \vee Q(a)$ are Horn. Finally, as $\mathcal{Q}_1$ has exponentially many atoms, we must restrict their occurrences in clauses. Accordingly, for $1 \leq i \leq 4$, define clauses of type Ci' to be of the form $C \vee E$ where C is of type Ci, $E \subseteq \pm\mathcal{Q}_1$, $\underline{C \vee E}$ is Horn and E has at most r negative literals ($\underline{C}$ is defined as before, hence it leaves atoms from $\mathcal{Q}_1$ unchanged). Now the $\mathcal{Q}$-splitting-replacement

strategy f first applies $\mathcal{Q}_1$-splitting as long as possible, then applies $\mathcal{Q}_0$-splitting as long as possible and then applies the replacement steps of Section 6 as long as possible. Succinct representations are now defined as: $S \sqsubseteq T$ iff for each $C \in S$, C is of type Ci' and satisfies Pi_T for some $1 \leq i \leq 4$. The abstract resolution procedure is defined as: $T \blacktriangleright T \cup \{B_1[\mathbf{x}_{r+1}] \vee -q_2 \vee \ldots \vee -q_k \sqcup D \sqcup E\} \cup \{q_i \vee B_i[\mathbf{x}_{r+1}] \mid 2 \leq i \leq k\}$ whenever $\mathsf{I}(T) \models_{\mathrm{p}} \underline{C}$, $C = B_1[\mathbf{x}_{i_1}] \sqcup \ldots \sqcup B_k[\mathbf{x}_{i_k}] \sqcup D \sqcup E$, $\underline{C}$ is Horn, $1 \leq i_1, \ldots, i_k \leq r$, B_1 is an ϵ-block, B_i is a negative ϵ-block and $q_i = \overline{B_i}$ for $2 \leq i \leq k$, $D \subseteq \pm\mathcal{Q}_0$ and $E \subseteq \pm\mathcal{Q}_1$ such that if $k = 1$ then E has at most r negative literals, and if $k > 1$ then E has no negative literal.

Lemma 10. *If $S \sqsubseteq T$ and $S \Rrightarrow_{\prec_s, f, \mathcal{R}} S'$ then $T \blacktriangleright^* T'$ for some T' such that $S' \sqsubseteq T'$.*

Now for deciding satisfiability of a set of flat and one-variable clauses we proceed as in the non-Horn case. But now instead of non-deterministically adding clauses, we compute a sequence $S = S_0 \blacktriangleright S_1 \blacktriangleright S_2 \ldots$ starting from the given set S, till no more clauses can be added, and then check whether $\Box$ has been generated. The length of this sequence is at most exponential. Computing S_{i+1} from S_i requires at most exponential time because the number of possibilities for C in the definition of $\blacktriangleright$ above is exponential. (Note that this idea of $\mathcal{Q}_1$-splitting would not have helped in the non-Horn case because we cannot bound the number of positive splitting literals in a clause in the non-Horn case, whereas Horn clauses by definition have at most one positive literal). Also note that APDS can be encoded using flat Horn clauses. Hence:

Theorem 6. *Satisfiability for the classes $\mathcal{CH}orn$ and $\mathcal{FH}orn$ is DEXPTIME-complete.*

Together with Theorem 1, this gives us optimal complexity for protocol verification:

Theorem 7. *Secrecy of cryptographic protocols with single blind copying, with bounded number of nonces but unbounded number of sessions is DEXPTIME-complete.*

8 Conclusion

We proved DEXPTIME-hardness of secrecy for cryptographic protocols with single blind copying, and improved the upper bound from 3-DEXPTIME to DEXPTIME. We improved the 3-DEXPTIME upper bound for satisfiability for the class $\mathcal{C}$ to NEXPTIME in the general case and DEXPTIME in the Horn case, which match known lower bounds. For this we invented new resolution techniques like ordered resolution with splitting modulo propositional reasoning, ordered literal replacements and decompositions of one-variable terms. As byproducts we obtained optimum complexity for several fragments of $\mathcal{C}$ involving flat and one-variable clauses. Security for several other decidable classes of protocols with unbounded number of sessions and bounded number of nonces is in DEXPTIME, suggesting that DEXPTIME is a reasonable complexity class for this class of protocols.

References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In *CSL'93*, pages 1–17. Springer-Verlag LNCS 832, 1993.
2. L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. North-Holland, 2001.
3. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.
4. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1), 1981.
5. H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 2004. To appear.
6. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *RTA'03*, pages 148–164. Springer-Verlag LNCS 2706, 2003.
7. H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. In *ESOP'03*, pages 99–113. Springer-Verlag LNCS 2618, 2003.
8. V. Cortier. *Vérification Automatique des Protocoles Cryptographiques*. PhD thesis, ENS Cachan, France, 2003.
9. N. A. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *FMSP'99*, Trento, Italy, 1999.
10. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *LICS'01*, pages 55–64. IEEE Computer Society Press, 2003.
11. J. Goubault-Larrecq, M. Roger, and K. N. Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 2004. To Appear. Available as Research Report LSV-04-7, LSV, ENS Cachan.
12. D. Monniaux. Abstracting cryptographic protocols with tree automata. In *SAS'99*, pages 149–163. Springer-Verlag LNCS 1694, 1999.
13. A. Riazanov and A. Voronkov. Splitting without backtracking. In *IJCAI'01*, pages 611–617, 2001.
14. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW'01*. IEEE Computer Society Press, 2001.
15. C. Weidenbach. Towards an automatic analysis of security protocols. In *CADE'99*, pages 378–382. Springer-Verlag LNAI 1632, 1999.