Iterative Specialisation of Horn Clauses

Christoffer Rosenkilde Nielsen, Flemming Nielson, and Hanne Riis Nielson

Technical University of Denmark
{crn,nielson,riis}@imm.dtu.dk

Abstract. We present a generic algorithm for solving Horn clauses through iterative specialisation. The algorithm is generic in the sense that it can be instantiated with any decidable fragment of Horn clauses, resulting in a solution scheme for general Horn clauses that guarantees soundness and termination, and furthermore, it presents sufficient criteria for completeness. We then demonstrate the use of the framework, by creating an instance of it, based on the decidable class $\mathcal{H}_1$, capable of solving a non-trivial protocol analysis problem based on the Yahalom protocol.

1 Introduction

Horn clauses have proven to be useful in many areas of computer science. They are very expressive (they are, in fact, Turing complete [1]) and yet they maintain a high clarity due to the simple format. This makes them attractive for many theoretical developments, as well as for practical purposes exemplified by the Prolog language.

Their usefulness has, however, also been restricted for the very same reasons that they are interesting: unrestricted Horn clauses are Turing complete. Thus Horn clause-based problems are often undecidable, and, similarly, Horn clause-based algorithms often have termination problems.

In this paper we shall show how to circumvent the first shortcoming, namely that problems formulated in unrestricted Horn clauses may be undecidable, thereby making Horn clauses more attractive for theoretical purposes such as static analysis. Specifically, we shall present a framework for finding, and iteratively improving the precision of, a model for any set of unrestricted Horn clauses. The framework is shown to guarantee soundness (i.e. it always returns a correct model) and termination, and it may in some cases also provide completeness (i.e. the model is the least model). Completeness depends on the chosen instance of the framework, as the framework is generic in the sense that it can be instantiated with any known decidable fragment of Horn clauses.

The general structure of our iterative framework is shown in Fig. 1. It is parameterised on a decidable fragment $\mathcal{H}$ of Horn clauses and given a formula φ the first step is to check whether φ is in $\mathcal{H}$, if so, then we can immediately construct its least model, denoted $\mathcal{N}(\varphi)$ on the figure, and we are done.

If φ is not in $\mathcal{H}$, then we apply an $\mathcal{H}$ -relaxation, $\mathcal{R}$, and the resulting formula, $\tilde{\varphi} = \mathcal{R}(\varphi)$, will be in $\mathcal{H}$. The relaxation guarantees that the least model of this

Fig. 1. Iterative scheme

formula (denoted $\hat{\varphi} = \mathcal{N}(\tilde{\varphi})$ on the figure) is also a model of φ , but it is possibly too large to be useful by itself. It may, however, contain useful information that can be used to *specialise* the original formula φ ; this operation is denoted $\varphi' = \mathcal{S}[\hat{\varphi}](\varphi)$ on Fig. 1. In order to compare the new and the original formula, we introduce a well-founded, *simplification ordering*, $\sqsubseteq$, over formulae; if the new formula is an improvement of the old, according to this ordering, then the above steps can be repeated. Otherwise the iteration stops and $\hat{\varphi}$ will be the resulting (approximative) model of φ .

The remainder of the paper is organised as follows. In Sect. 2 we give the background in Horn clauses necessary for understanding the development. We then, formally, present the iterative scheme in Sect. 3 and show that it satisfies several attractive properties. To illustrate its usefulness, we present an instance of the scheme in Sect. 4, using the decidable fragment of Horn clauses $\mathcal{H}_1$, and afterwards in Sect. 5 demonstrate how this instance can be used for verifying cryptographic protocols, by applying it to a specific analysis problem based on one of the well-known protocols of the literature; the Yahalom key-distribution protocol. Finally, Sect. 6 reflects and concludes.

2 Horn Clauses

The standard syntax of Horn clauses is presented in Table 1 and should be read as follows. A Horn *formula* $\varphi \in \mathbf{HC}$ is a finite set of implications, usually referred to as *clauses*. Every clause c is on the form $g_0 \Leftarrow g_1, \dots, g_l$ where the literals g_0 and $g_1, \dots, g_l$ are the head and the precondition of c , respectively. A *literal* is of the form $\mathbf{p}(t_1, \dots, t_m)$ where $\mathbf{p}$ is an m -ary predicate symbol and t_i is a term built up from variables (indicated by a capitalised first letter) through constructor applications. In the following, we shall sometimes refer to constructors of arity 0 as *constants*, and clauses without preconditions as *facts*, written without the implication arrow.

Table 1. Horn syntax

$\varphi ::= \{c_1, \dots, c_k\}$	$g ::= p(t_1, \dots, t_m)$
$c ::= g_0 \Leftarrow g_1, \dots, g_l$	$t ::= X \mid f(t_1, \dots, t_n)$

We let expressions, e , range over formulae, clauses, literals and terms, and define $\text{Var}(e)$, $\text{Con}(e)$, and $\text{Pred}(e)$, to be the set of variable, constructor and predicate symbols, respectively, that occur in e ¹. A term, t , is called *ground* if it consists of no variables, i.e. $\text{Var}(t) = \emptyset$, and for a given formula, φ , the set of all ground terms built from $\text{Con}(\varphi)$ is called its *Herbrand universe*, denoted H_φ .

Horn formulae are interpreted relative to an interpretation $\rho \in \mathbf{R}$ that maps predicate symbols to corresponding term relations; i.e. an m -ary predicate symbol is mapped to an m -ary term relation which is a subset of H_φ^m . Letting θ denote a substitution mapping variables to ground terms, we then define a satisfaction relation $\models$ as presented² in Table 2.

Table 2. Satisfaction relation

$\rho \models \varphi$	iff	$\forall c \in \varphi : \rho \models c$
$\rho \models g_0 \Leftarrow g_1, \dots, g_l$	iff	$\forall \theta : \bigwedge_{i=1}^l (\rho, \theta \models g_i) \Rightarrow (\rho, \theta \models g_0)$
$\rho, \theta \models p(t_1, \dots, t_m)$	iff	$\langle t_1\theta, \dots, t_m\theta \rangle \in \rho(p)$

An interpretation ρ , that satisfies a Horn formula φ is called a model of φ . We then state the following well-known result [2]:

Proposition 1 (Least model). *The set of models of a Horn formula $\varphi \in \mathbf{HC}$ constitutes a Moore family (i.e. it is closed under intersection), and thus it has a unique least model: $\rho_\varphi = \bigcap \{\rho \mid \rho \models \varphi\}$.*

Given a set of Horn clauses we usually want to answer questions such as *membership* (does $\langle t_1, \dots, t_m \rangle$ belong to the least model of $\mathbf{p}$) or *non-emptiness* (is $\mathbf{p}$ mapped to a non-empty set in the least model). Unfortunately, unrestricted Horn clauses are Turing complete [1], and these questions may be undecidable. Therefore we are particularly interested in subsets of Horn clauses where they are decidable, and we shall refer to these fragments as *decidable classes*.

All our questions concern the least model of the formula, but as this model may include infinite sets, we need a finite representation. This representation can itself be expressed by Horn clauses, but now in a form that allows membership in the least model to be determined in a straightforward manner (i.e. linear time).

¹ Variables occurring in a clause are implicitly universally quantified in that clause; thus, without loss of generality, one may assume that variables in different clauses are α -renamed apart.

² We adopt the usual postfix notation for substitutions, i.e. $\theta\theta'$ stands for $\theta' \circ \theta$; for an expression e then $e\theta$ represents the expression resulting from replacing all variables $X \in \text{dom}(\theta)$ in e with their corresponding ground terms. When convenient, we shall use the notation $[X \mapsto t]$ for the substitution mapping only the variable X to the ground term t .

Table 3. A normal form for Horn clauses

(N1)	$p(X_1, \dots, X_m)$	$\Leftarrow$	$q_1(X_1), \dots, q_m(X_m)$	,	$\forall_{i \neq j} X_i \neq X_j \wedge m \geq 2$
(N2)	$p(f(X_1, \dots, X_n))$	$\Leftarrow$	$q_1(X_1), \dots, q_n(X_n)$	,	$\forall_{i \neq j} X_i \neq X_j$

Such a form is called a *normal form*, and a transformation from a decidable class into a normal form is referred to as a *normalisation*, when it preserves the set of models from the original formula and can be performed through a finite number of operations. We present an example of a normal form in Table 3; notice that we require the heads to be *linear*, i.e. no variable occurs twice. This normal form essentially describes context-free grammars in a succinct manner and provides a linear time lookup for both membership and non-emptiness.

To ease the understanding of the development in the following sections, the reader may think of this particular normal form whenever we refer to an unspecified normal form for Horn clauses, bearing in mind, of course, that the results still apply to any other normal form as well. We shall write $\widehat{\varphi} \in \mathbf{HC}$ to denote a formula on normal form.

3 Iterative Scheme

The general idea in the iterative framework was already explained in Sect. 1. We shall now present the key ingredients that, in addition to the normalisation introduced above, include the relaxation mechanism, the specialisation function and the simplification ordering.

3.1 The Refinement Scheme

Recall first that $\mathbf{HC}$ and $\widehat{\mathbf{HC}}$ denote the (infinite) sets of Horn formulae and normalised Horn formulae, respectively. We then define the premise of our intuition as follows: there exists a function that takes a general Horn formula and returns an over-approximative formula (ie. a formula, for which all models are also models of the original formula, and thus it has a weakly larger least model) which belongs to some decidable class $\mathcal{H}$. We shall call such a function an $\mathcal{H}$ -relaxation and formally define it as follows:

Definition 1 ($\mathcal{H}$ -Relaxation). Let $\mathcal{H} \subseteq \mathbf{HC}$, then a function $\mathcal{R} : \mathbf{HC} \rightarrow \mathcal{H}$ is an $\mathcal{H}$ -relaxation if:

$$\forall \varphi : \forall \rho : \quad \rho \models \mathcal{R}(\varphi) \Rightarrow \rho \models \varphi$$

As $\mathcal{H}$ is decidable, then there exists a function that takes a formula $\varphi \in \mathcal{H}$ and returns a normalised formula that represents its unique least model ρ_φ . Such a function is usually called an $\mathcal{H}$ -normalisation:

Definition 2 ($\mathcal{H}$ -Normalisation). Let $\mathcal{H} \subseteq \mathbf{HC}$, then a function $\mathcal{N} : \mathcal{H} \rightarrow \widehat{\mathbf{HC}}$ is an $\mathcal{H}$ -normalisation if:

$$\forall \varphi \in \mathcal{H} : \forall \mathbf{p} \in \text{Pred}(\varphi) : \quad t \in \rho_\varphi(\mathbf{p}) \Leftrightarrow t \in \rho_{\mathcal{N}(\varphi)}(\mathbf{p})$$

Notice that the definition allows the normalised formula to also include auxiliary predicates.

The combination of the relaxation and the normalisation establishes that we can always find some model of a formula. However, we are usually only interested in the least model or a model close to the least model. Naturally, if $\mathcal{R}$ returns a formula with the same least model as the original, then the result is precise and we are done. But this is usually not the case, and often the approximation will be too coarse to be useful for the intended purpose. In this case, we may inspect this coarse model and look for information that allow us to produce a safe transformation of the formula, that may produce better results. This is called a specialisation.

Definition 3 (Specialisation). *A specialisation is a higher-order function $\mathcal{S} : \widehat{\mathbf{HC}} \rightarrow \mathbf{HC} \rightarrow \mathbf{HC}$ that satisfies:*

$$\forall \widehat{\varphi} : \forall \varphi : \forall \rho \subseteq \rho_{\widehat{\varphi}} : \quad \rho_{\widehat{\varphi}} \models \varphi \Rightarrow (\rho \models \mathcal{S}[\widehat{\varphi}](\varphi) \Leftrightarrow \rho \models \varphi)$$

A specialisation takes an approximative normal formula $\widehat{\varphi}$ and specialises the formula φ , such that the specialisation $\mathcal{S}[\widehat{\varphi}](\varphi)$ maintains the set of models, smaller than or equal to the $\rho_{\widehat{\varphi}}$.

The fourth, and last, ingredient we shall require for the iterative scheme, is an order that ensures termination.

Definition 4 (Simplification ordering). *A relation $\sqsubseteq : \mathbf{HC} \times \mathbf{HC}$ is a simplification ordering if it is a well-founded, partial order.*

When convenient, we shall use familiar notation $\sqsupseteq$ for the inverse of $\sqsubseteq$ and likewise $\sqsubset$ and $\sqsubset$ for their respective strict counterparts.

The combination of these operators is then supposed to form the basis of the iterative scheme, for finding solutions for unrestricted Horn problems. However, for this to be achieved, we shall furthermore require them to form an $\mathcal{H}$ -refinement scheme:

Definition 5 ($\mathcal{H}$ -Refinement Scheme). *A quadruple $(\mathcal{R}, \mathcal{N}, \mathcal{S}, \sqsubseteq)$; consisting of an $\mathcal{H}$ -relaxation, an $\mathcal{H}$ -normalisation, a specialisation, and a simplification ordering, is an $\mathcal{H}$ -refinement scheme if:*

$$\forall \varphi : \forall \widehat{\varphi} : \forall \mathbf{p} \in \text{Pred}(\varphi) : \quad \widehat{\varphi} = \mathcal{N}(\mathcal{R}(\varphi)) \Rightarrow (\rho_{\widehat{\varphi}}(\mathbf{p}) \sqsupseteq \rho_{\mathcal{S}[\widehat{\varphi}](\varphi)}(\mathbf{p}))$$

3.2 The Iterative Scheme

The refinement scheme forms the basis of the larger iterative scheme, that was graphically presented in Fig. 1 in Sect. 1. Essentially, for any decidable class $\mathcal{H}$ for which there exists an $\mathcal{H}$ -refinement scheme, $(\mathcal{R}, \mathcal{N}, \mathcal{S}, \sqsubseteq)$, the iterative scheme will find a best solution to any unrestricted Horn problem.

The algorithm can terminate in two ways: (1) Either the scheme eventually produces a specialisation within $\mathcal{H}$, which is therefore decidable, and the result is the least model of the input formula; or (2) the scheme eventually fails to improve the formula further, with respect to $\sqsubseteq$, and the result is then the best known model of the input formula.

3.3 Properties of the Scheme

As part of justifying the proposed scheme, we shall show that it satisfies three key properties: we shall prove that the algorithm always terminates, that the normalised output represents a model of the input formula, and that if the algorithm produces a formula within the decidable class, then the corresponding normal form represents the least model of the original input formula.

First we shall give the termination result.

Theorem 1 (Termination). *The algorithm will always terminate.*

Proof. Assuming that the functions $\mathcal{R}$, $\mathcal{N}$, and $\mathcal{S}$ are terminating, then the result follows directly from the fact that $\sqsubseteq$ is well-founded. $\square$

The next result states that the algorithm is sound and justifies the iterative approach.

Theorem 2 (Soundness). *The algorithm will always produce a model of the input formula φ . Each iteration (weakly) improves precision of this model.*

Proof. Notice that, if the algorithm terminates in the i th iteration, then the result is either $\mathcal{N}(\varphi_i)$ or $\mathcal{N}(\mathcal{R}(\varphi_i))$. Thus, the proof amounts to show that these normalised clauses represent models of the input formula for all i and that each iteration retains the set of models smaller than or equal to found model ; i.e. that $\forall \rho \subseteq \rho_{\widehat{\varphi}_i} : (\rho \models \varphi_i \Leftrightarrow \rho \models \varphi)$. This is proven by induction.

For $i = 0$ the result holds trivially. For the iterative step, let $\widehat{\varphi}_i = \mathcal{N}(\mathcal{R}(\varphi_i))$ such that $\rho_{\widehat{\varphi}_i} \models \varphi_i$, and assume that $\forall \rho \subseteq \rho_{\widehat{\varphi}_i} : (\rho \models \varphi_i \Leftrightarrow \rho \models \varphi)$. Next, let $\varphi_{i+1} = \mathcal{S}[\widehat{\varphi}_i](\varphi_i)$ and $\widehat{\varphi}_{i+1} = \mathcal{N}(\mathcal{R}(\varphi_{i+1}))$. Now, according to the definition of a relaxation and normalisation, this means that $\rho_{\widehat{\varphi}_{i+1}} \models \varphi_{i+1}$. Furthermore, by the definition of a refinement scheme, we have $\forall \rho : \forall \mathbf{p} \in \text{Pred}(\varphi_i) : \rho_{\widehat{\varphi}_i}(\mathbf{p}) \supseteq \rho_{\mathcal{R}(\varphi_{i+1})}(\mathbf{p})$, and thus, by definition of the normalisation, $\forall \rho : \forall \mathbf{p} \in \text{Pred}(\varphi_i) : \rho_{\widehat{\varphi}_i}(\mathbf{p}) \supseteq \rho_{\widehat{\varphi}_{i+1}}(\mathbf{p})$. Hence, by definition of a specialisation and the assumption, and as the specialisation does not introduce any new predicates, we have that $\forall \rho \subseteq \rho_{\widehat{\varphi}_{i+1}} : (\rho \models \varphi_{i+1} \Leftrightarrow \rho \models \varphi_i \Leftrightarrow \rho \models \varphi)$. $\square$

And lastly we have the partial completeness result:

Theorem 3 (Partial Completeness). *If the algorithm terminates in (1), then it produces the least model of the input formula φ .*

Proof. The result follows directly from the proof for Theorem 2. $\square$

4 Application to $\mathcal{H}_1$

This section presents one instance of the iterative framework, or more precisely, a specific refinement scheme. For this, we have chosen the decidable subclass $\mathcal{H}_1$, originally introduced in [3]. $\mathcal{H}_1$ describes strongly recognisable relations; i.e. finite unions of Cartesian products of recognisable tree languages. In fact, every clause in $\mathcal{H}_1$ is normalisable (to the normal form presented in Sect. 2) and the equivalent normal form can be constructed in deterministic exponential time [3].

Previously, this class has proven very useful for static analysis purposes, both for specifying a Control-Flow Analysis of Spi [3] as well as for verifying real implementations of cryptographic protocols in the C language [4]. Yet, as we shall see in Sect. 5.1, a direct attempt at specifying protocol analysis, results in clauses outside this class. This motivates the use of the iterative scheme of Sect. 3.

We begin by briefly introducing the $\mathcal{H}_1$ -class itself, whereafter we will define the operators in the $\mathcal{H}_1$ -refinement scheme, $(r, n, s, \leq)$, needed for using the iterative scheme of Fig. 1.

4.1 The Class $\mathcal{H}_1$

We shall say the two variables X and Y are *connected* within a set of literals $\{g_1, \dots, g_k\}$ if there exists a sequence of literals $g_{p_1} \dots g_{p_l}$ with $l \geq 1$ and $g_{p_i} \in \{g_1, \dots, g_k\}$ such that $X \in \text{Var}(g_{p_1})$, $Y \in \text{Var}(g_{p_l})$ and $\text{Var}(g_{p_i}) \cap \text{Var}(g_{p_{i+1}}) \neq \emptyset$ for all $1 \leq i \leq l - 1$.

We then say that the clause $g_0 \Leftarrow g_1, \dots, g_k$ has the property H1 if it satisfies the requirements given in Table 4. Here we call two variables *siblings* in a literal or term, if they occur as arguments of a common parent; i.e. X, Y are siblings in $p(X, Y)$ and $p(Z, f(X, Y))$ but not in $p(X, f(Y))$.

Table 4. Property H1 for clauses of the form $g_0 \Leftarrow g_1, \dots, g_k$

(H1.1)	g_0 is linear.
(H1.2)	If $X, Y \in \text{Var}(g_0)$ are connected in $\{g_1, \dots, g_k\}$, then they are siblings in g_0 .

A formula φ belongs to the class $\mathcal{H}_1$ if all clauses in φ have property H1.

Example 1. Consider, as a small, running example, the following clause, borrowed from the protocol analysis case study in Sect. 5.1:

$$\text{net}(\langle X_{enc}, \{X_{nb}\}_{X_{kab}} \rangle) \Leftarrow \text{net}(\langle \{ \langle b, X_{kab}, n_a, X_{nb} \rangle \}_{k_{as}}, X_{enc} \rangle)$$

Here $\langle t_1, \dots, t_n \rangle$ is an n -ary constructor and $\{t_1\}_{t_2}$ a binary constructor.

This clause is *not* in $\mathcal{H}_1$ as the variables X_{enc} and X_{nb} (and similarly X_{enc} and X_{kab}) are connected in the precondition but are not siblings in the head, thus violating the rule (H1.2) in Table 4.

4.2 Relaxation Operator

We now want to define a relaxation function, that always produces clauses within $\mathcal{H}_1$. In [5] it is shown how general Horn clauses can be approximated by $\mathcal{H}_1$ clauses. We shall follow this approach, but slightly improve it, as the relaxation operator we suggest, for clauses violating (H1.2), is more precise as compared to the approximation technique suggested in [5].

Let e be an expression. We then write $e[X \rightsquigarrow t]$ for the expression that is as e , except that the leftmost occurrence of X is replaced by t . We shall also employ

the shorthand notation α for a sequence of literals $g_1, \dots, g_l$ when convenient, and point-wise extend substitutions and $\models$ for this. We then define a function $r(\varphi) = \bigcup \{r(c) \mid c \in \varphi\}$ as follows:

$$r(g \Leftarrow \alpha) = \begin{cases} r(g[X \rightsquigarrow X'] \Leftarrow \alpha[X \mapsto X'], \alpha) & \begin{array}{l} (a) \text{ if } g \Leftarrow \alpha \text{ violates (H1.1)} \\ \text{and where } X' \text{ is fresh} \end{array} \\ r(g \Leftarrow \alpha[X \mapsto X'], \mathbf{p}(X, Z_1, \dots, Z_k)) \cup \{\mathbf{p}(X, Z_1, \dots, Z_k) \Leftarrow \alpha\} & \begin{array}{l} (b) \text{ if } g \Leftarrow \alpha \text{ violates (H1.2)} \\ \text{and where } X' \text{ and } \mathbf{p} \text{ are fresh} \end{array} \\ \{g \Leftarrow \alpha\} & (c) \text{ otherwise} \end{cases}$$

In both (a) and (b) X is the leftmost variable that gives rise to the violation, and in (b) the siblings $Z_1, \dots, Z_k$ of X in g are carried on to the auxiliary predicate, to retain the highest amount of precision.

From the definition it is apparent that the set of clauses generated by r will always be in $\mathcal{H}_1$, and that, if applied to a set of clauses already within $\mathcal{H}_1$, r will be the identity function. Furthermore, as the set of variables in a clause is always finite, it should be fairly obvious that r will always terminate.

Now, prior to showing that r indeed constitutes a relaxation function, we shall give an auxiliary result that benefits the presentation of the proof.

Fact 4. If $\rho \models c$ then $\forall X, X' : \rho \models c[X \mapsto X']$.

Which allows us to establish the following result.

Lemma 1. *The function r is a relaxation operator.*

Proof. The proof proceeds by induction in the derivation sequence establishing $r(\varphi)$:

- (a), remembering that $Y \notin \text{Var}(g \Leftarrow \alpha)$ the result follows from Fact 4 :

$$\begin{aligned} & \rho \models g[X \rightsquigarrow X'] \Leftarrow \alpha[X \mapsto X'], \alpha \\ \Rightarrow & \rho \models (g[X \rightsquigarrow X'] \Leftarrow \alpha[X \mapsto X'], \alpha)[X' \mapsto X] \\ \Rightarrow & \rho \models g \Leftarrow \alpha, \alpha \\ \Rightarrow & \rho \models g \Leftarrow \alpha \end{aligned}$$

- (b) is shown analogously, relying on the fact that the auxiliary predicate $\mathbf{p}$ does not occur in the head of any other clause and thus the precondition in the first clause can only be satisfied if the second clause is satisfied.
- (c) holds vacuously. □

Example 2 (Example 1 continued). Recall that the clause from Example 1 was not in $\mathcal{H}_1$, as the variables X_{enc} and X_{nb} violated rule (H1.2). Thus, the relaxation r applied to this clause will decouple these variables, resulting in the following two clauses, now both in $\mathcal{H}_1$.

$$\left\{ \begin{array}{l} \text{net}(\langle X_{enc}, \{X_{nb}\}_{X_{kab}} \rangle) \Leftarrow \text{net}(\langle \{ \langle b, X_{kab}, n_a, X_{nb} \rangle \}_{k_{as}}, X' \rangle), \mathbf{p}(X_{enc}), \\ \mathbf{p}(X_{enc}) \Leftarrow \text{net}(\langle \{ \langle b, X_{kab}, n_a, X_{nb} \rangle \}_{k_{as}}, X_{enc} \rangle) \end{array} \right\}$$

4.3 $\mathcal{H}_1$ -Normalisation

Normalising a set of $\mathcal{H}_1$ -clauses amounts to bringing the clauses onto the normal form given in Table 3. This is a well-known procedure that was initially described in a direct manner in [3] and later using resolution techniques in [5]. Basically the procedure consists of iteratively extending the set of normal clauses by simplifying non-normal clauses using the current set of normal clauses, this procedure is continued until no further simplification can be performed; then the non-normal clauses are redundant and can be removed.

4.4 Specialisation Operator

The specialisation is supposed to utilise information in a given normal form approximation $\widehat{\varphi}$ of a formula φ , to produce a new formula φ' , that maintains the set of models smaller than or equal to $\rho_{\widehat{\varphi}}$, but may yield smaller approximations. One approach to define such an operator, is to define a function that safely eliminates variables from the formula.

First we shall require a formulation of which ground terms a variable may be substituted by in a clause.

Definition 6 (Substitution set). *The substitution set for $X \in \text{Var}(\varphi)$ is given by $\mathcal{T}_{\varphi}(X) = \{X\theta \mid \bigwedge_{i=0}^l \rho_{\varphi}, \theta \models g_i \wedge (g_0 \Leftarrow g_1, \dots, g_l) \in \varphi\}$.*

Unfortunately, some of these substitution sets may be infinite and we are therefore particularly interested in the set $\mathcal{F}_{\varphi}$ of variables $X \in \text{Var}(\varphi)$ for which $\mathcal{T}_{\varphi}(X)$ is finite. However, it may not always be feasible to determine the complete set $\mathcal{F}_{\varphi}$, or the corresponding substitution sets, and thus we shall say that a *finiteness substitution mapping* I_{φ} is permissable if $\text{dom}(I_{\varphi}) \subseteq \mathcal{F}_{\varphi}$ and $\forall X \in \text{dom}(I_{\varphi}) : I_{\varphi}(X) \supseteq \mathcal{T}_{\varphi}(X)$.

As neither the relaxation nor the normalisation renames variables, and as $\rho_{\widehat{\varphi}} \models \varphi$, it follows that for all $X \in \text{Var}(\varphi)$ then $\mathcal{T}_{\varphi}(X) \subseteq \mathcal{T}_{\widehat{\varphi}}(X)$. Thus a permissable $I_{\widehat{\varphi}}$ is also a permissable I_{φ} , and the idea is then use the former to perform a *complete expansion* of the formula. This is done by the function, $s : \widehat{\mathbf{HC}} \rightarrow \mathbf{HC} \rightarrow \mathbf{HC}$, defined as follows:

$$s[\widehat{\varphi}](\varphi) = \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \forall X \in \text{dom}(\theta) : X\theta \in I_{\widehat{\varphi}}(X) \}$$

Note that $s[\widehat{\varphi}]$ is the identity function if there are no variables in φ with finite substitution sets in $I_{\widehat{\varphi}}$.

Lemma 2. *The function s is a specialisation function.*

Proof. Assume $\widehat{\varphi}$, φ and ρ , such that $\rho \subseteq \rho_{\widehat{\varphi}}$. We then have

$$\begin{aligned} \rho &= s[\widehat{\varphi}](\varphi) \\ \Leftrightarrow \rho &= \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \forall X \in \text{dom}(\theta) : X\theta \in I_{\widehat{\varphi}}(X) \} \\ \Leftrightarrow \rho &= \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \forall X \in \text{dom}(\theta) : X\theta \in \mathcal{T}_{\widehat{\varphi}}(X) \} \\ \Leftrightarrow \rho &= \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \forall (g_0 \Leftarrow g_1, \dots, g_l) \in \varphi : \bigwedge_{i=0}^l \rho_{\widehat{\varphi}} \models g_i\theta \} \\ \Leftrightarrow \rho &= \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \forall (g_0 \Leftarrow g_1, \dots, g_l) \in \varphi : \bigwedge_{i=0}^l \rho \models g_i\theta \} \\ \Leftrightarrow \rho &= \bigcup \{ \varphi\theta \mid \text{dom}(\theta) = \text{dom}(I_{\widehat{\varphi}}) \wedge \rho \models \varphi\theta \} \\ \Leftrightarrow \rho &= \varphi \end{aligned}$$

In the second and fourth step we use that $\rho \subseteq \rho_{\widehat{\varphi}}$ □

It then only remains to show how $I_{\widehat{\varphi}}$ can be determined for $\mathcal{H}_1$, or more specifically, for the normal form in Table 3. Such a mapping can be obtained by the following inductively defined procedure:

$$\boxed{
 \begin{array}{c}
 \frac{\forall(\mathbf{p}(t) \Leftarrow \alpha) \in \widehat{\varphi} : \text{Pred}(\alpha) \subseteq \text{dom}(F)}{\mathbf{p} \in \text{dom}(F) \quad \wedge} \\
 \forall(\mathbf{p}(f(X_1, \dots, X_n)) \Leftarrow \mathbf{q}_1(X_1), \dots, \mathbf{q}_n(X_n)) \in \widehat{\varphi} : \\
 \quad \forall(t_1, \dots, t_n) \in F(\mathbf{q}_1) \times \dots \times F(\mathbf{q}_n) : \\
 \quad \quad f(t_1, \dots, t_n) \in F(\mathbf{p}) \\
 \\
 \frac{\forall(g \Leftarrow \alpha) \in \widehat{\varphi} : \mathbf{p}(X) \in \alpha \Rightarrow \mathbf{p} \in \text{dom}(F)}{X \in \text{dom}(I_{\widehat{\varphi}}) \quad \wedge \quad \forall(g \Leftarrow \alpha) \in \widehat{\varphi} : \mathbf{p}(X) \in \alpha \Rightarrow F(\mathbf{p}) \subseteq I_{\widehat{\varphi}}(X)}
 \end{array}
 }$$

The mapping is built by first finding a mapping F , of the unary predicates with a finite mapping in $\rho_{\widehat{\varphi}}$, and then use this to build $I_{\widehat{\varphi}}$.

Example 3 (Example 1 continued). Assume the finiteness substitution mapping $I_{\widehat{\varphi}} = [X_{n_b} \mapsto \{n_b\}, X_{k_{ab}} \mapsto \{k_{ab}, m_{\bullet}\}]$, then a complete expansion of the clause from Example 1 is:

$$\begin{aligned}
 &\{ \text{net}(\langle X_{enc}, \{n_b\}_{k_{ab}} \rangle) \Leftarrow \text{net}(\langle \{b, k_{ab}, n_a, n_b\}_{k_{as}}, X_{enc} \rangle), \\
 &\quad \text{net}(\langle X_{enc}, \{n_b\}_{m_{\bullet}} \rangle) \Leftarrow \text{net}(\langle \{b, m_{\bullet}, n_a, n_b\}_{k_{as}}, X_{enc} \rangle) \}
 \end{aligned}$$

4.5 Simplification Ordering

As the specialisation expands the clauses of the formula, an obvious choice of simplification ordering is an ordering where clauses are greater than their expanded counterpart.

$$\forall \varphi : \forall \varphi' : \quad \varphi \leq \varphi' \Leftrightarrow (\exists \widehat{\varphi} : \varphi = s[\widehat{\varphi}](\varphi'))$$

It follows that equivalence resolves to equality, and thus the check in 4 in the iterative scheme will amount to the test for identity. We then have the result:

Lemma 3. *The operator $\leq$ is a simplification ordering.*

Proof. We have trivially that $\leq$ is a partial order. That it is also well-founded follows from the fact that any formula φ can only hold a finite set of variables. As a complete expansion (modulo identity) will only expand each clause into a finite set of new clauses, and each expanded clause will have a strictly lower number of variables, then any formula can only be completely expanded a finite number of times. $\square$

4.6 $\mathcal{H}_1$ -Refinement Scheme

Finally, to show that the different ingredients work together as intended, we must show that they form an $\mathcal{H}_1$ -refinement scheme.

Lemma 4. *The quadruple $(r, n, s, \leq)$ forms an $\mathcal{H}_1$ -refinement scheme.*

Proof. To prove this, we must show that for all t and for all $\mathbf{p} \in \text{Pred}(\varphi)$, then $t \in \rho_{r(s[\widehat{\varphi}])(\varphi)}(\mathbf{p})$ implies $t \in \rho_{\widehat{\varphi}}(\mathbf{p})$, when $\widehat{\varphi} = n(r(\varphi))$. But as the specialisation only eliminates variables, and the relaxation only decouples variables, we have that $t \in \rho_{r(s[\widehat{\varphi}])(\varphi)}(\mathbf{p})$ implies $t \in \rho_{s[\widehat{\varphi}](r(\varphi))}(\mathbf{p})$. But then the result follows directly, as $\rho_{r(\varphi)} = \rho_{n(r(\varphi))} = \rho_{\widehat{\varphi}}$, and thus $t \in \rho_{s[\widehat{\varphi}](r(\varphi))}(\mathbf{p})$ implies $t \in \rho_{\widehat{\varphi}}(\mathbf{p})$ by definition of the specialisation, which gives the result we seek. $\square$

5 Worked Example

In this section, we shall give an example of how Horn clauses can be used for formalising some analysis problems in a succinct and clear manner. But, as we shall also see, such specifications easily end up outside any known decidable classes, motivating the use of the iterative scheme.

We have chosen our example from protocol analysis, more specifically the validation of the key-distribution protocol Yahalom [6], which is given in the classical Alice-Bob notation as follows:

1. $A \rightarrow B$: A, N_A
2. $B \rightarrow S$: $B, \{A, N_A, N_B\}_{K_{BS}}$
3. $S \rightarrow A$: $\{B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
4. $A \rightarrow B$: $\{A, K_{AB}\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Yahalom is considered a secure protocol (cf. [6,7]), but it is an interesting case study, because it proves troublesome for many analyses. In particular, independent attribute analyses, such as [8], will yield false positives for Yahalom, in the presence of several principals.

5.1 Modelling Protocols

Modelling protocol narrations in Horn clauses is relatively straightforward, as Horn clauses provide the means for an almost direct translation from the Alice-Bob notation. One such translation scheme was presented intuitively in [9] and refined in [10], and we shall draw inspiration from these approaches.

We assume that all messages are sent on one global network, and that all principals have access to this network. This is supposed to be a realistic model of the Internet, without adding an extra layer of security to the protocol, and we will describe it through the predicate *net*. Secondly, we shall model encryptions and tuples through constructors, but to ease readability, we will employ the familiar notation $\{t_1\}_{t_2}$ for the binary encryption constructor, and $\langle t_1, \dots, t_n \rangle$ for the n -ary tupling constructor.

The result of this method of translation, for one instance of Yahalom, is given in Table 5. Here we have adopted the notation $g, g' \Leftarrow \alpha$ as a shorthand for the clauses $g \Leftarrow \alpha$ and $g' \Leftarrow \alpha$. To stress the intent of the protocol, we add predicates on the form *p*.key for recording the ground terms that principal p binds and believes to be the distributed key; e.g. k_{ab} .

Each step of the protocol is translated into one clause, where the right-hand side represents the requirements imposed by the sender, and the left-hand side

Table 5. Yahalom in Horn clauses

1.	$\text{net}(\langle a, n_a \rangle)$
2.	$\text{net}(\langle b, \{ \langle X_a, X_{n_a}, n_b \rangle \}_{k_{bs}} \rangle) \Leftarrow \text{net}(\langle X_a, X_{n_a} \rangle)$
3.	$\text{net}(\langle \{ \langle b, k_{ab}, X_{n_a}, X_{n_b} \rangle \}_{k_{as}}, \{ \langle a, k_{ab} \rangle \}_{k_{bs}} \rangle) \Leftarrow \text{net}(\langle b, \{ \langle a, X_{n_a}, X_{n_b} \rangle \}_{k_{bs}} \rangle)$
4.	$\text{net}(\langle X_{enc}, \{ X_{n_b} \}_{X_{k_{ab}}} \rangle), \text{a_key}(X_{k_{ab}}) \Leftarrow \text{net}(\langle \{ \langle b, X_{k_{ab}}, n_a, X_{n_b} \rangle \}_{k_{as}}, X_{enc} \rangle)$
5.	$\text{b_key}(X_{k_{ab}}) \Leftarrow \text{net}(\langle \{ \langle a, X_{k_{ab}} \rangle \}_{k_{bs}}, \{ n_b \}_{X_{k_{ab}}} \rangle)$

the message sent upon the network in that particular step. In addition to the four steps of the protocol, a fifth step is added to reflect the reception of the fourth message. The modelling captures the basic assumption of the protocol, namely that the initiator a and the server s both know the two other principals on advance, but that the responder b does not necessarily need to initially know (or trust) a .

Notice that the specification allows a relational analysis as the relationship between variables in each message is remembered. As mentioned above, this is required for correctly validating Yahalom. In this example, we have, for simplicity, chosen to model only one instance of the protocol, but the methodology could be extended to also model multiple instances between multiple principals, maintaining the relational property, by using standard approximation techniques. This is, however, beyond the scope of this paper.

5.2 Network Attacker and Analysis

After modelling how the legitimate principals of the system interact, we usually want to apply an analysis for ensuring that the protocol cannot be compromised by one or more malicious principals.

One method, that has been employed successfully numerous times before, is modelling the so-called Dolev-Yao attacker [11], also known as the hardest network attacker [12], and investigate whether various security properties are upheld in presence of this attacker. In the Dolev-Yao model, the capabilities of this attacker are defined to be: (1) receive and intercept all messages sent on the global network and send new messages onto the global network; (2) decrypt encrypted messages, if it knows the encryption key, and construct new encryptions from known terms; (3) decompose and compose tuples; and (4) generate new ground terms. The attacker can, in principle, compose and decompose tuples of any arity. However, in [8] it is shown that restricting the attacker to work on a limited set of arities, does not limit its capabilities, as long as the set includes all arities occurring in the protocol and at least one additional arity. This means that a Dolev-Yao attacker, for a specific protocol, is directly translatable into Horn clauses. If we let $\mathcal{K}$ denote the set of occurring arities (for Yahalom this set is $\{2, 3, 4\}$), we can create an extended set $\mathcal{K}_+ = \mathcal{K} \cup \{1\}$, as tuples of arity 1 never occurs, and a generic Dolev-Yao attacker can then be formulated as in Table 6.

The attacker is described through a predicate dy , storing its accumulated knowledge. The first three rules are then straightforward, and the rule (DY4)

Table 6. Dolev-Yao attacker in Horn clauses

(DY1)	$\text{dy}(X) \Leftarrow \text{net}(X)$ $\text{net}(X) \Leftarrow \text{dy}(X)$	
(DY2)	$\text{dy}(X) \Leftarrow \text{dy}(\{X\}_{X_{key}}), \text{dy}(X_{key})$ $\text{dy}(\{X\}_{X_{key}}) \Leftarrow \text{dy}(X), \text{dy}(X_{key})$	
(DY3)	$\text{dy}(X_1), \dots, \text{dy}(X_k) \Leftarrow \text{dy}(\langle X_1, \dots, X_k \rangle)$ $\text{dy}(\langle X_1, \dots, X_k \rangle) \Leftarrow \text{dy}(X_1), \dots, \text{dy}(X_k)$	} if $k \in \mathcal{K}_+$
(DY4)	$\text{dy}(m_\bullet)$	

expresses that the attacker may generate arbitrary ground terms, but, in the modelling, all of these are coalesced into one equivalence class represented by the canonical name $m_\bullet$. Note, that a direct consequence of the rule (DY1) is that the attacker's knowledge and the net are identical.

A conjunction of a protocol description with the attacker in Table 6 constitutes an instance of an analysis problem, and the least model of such a problem allows us to investigate certain properties of the protocol. In its most simple form, such as the formula in Table 5, the protocol descriptions allow direct verification of confidentiality (and integrity through the predicates `a_key` and `b_key`) but many other properties can be verified as well using annotation techniques [8].

5.3 Results

Even small analysis problems, such as the one presented in this section, falls, to the best of our knowledge, outside the decidable fragments of Horn Clauses. Hence we must rely on the iterative scheme to determine a best model of our analysis problem.

For this, we have implemented the iterative scheme in OCaml³. The result is an extremely succinct implementation; the normalisation itself is constrained to 110 lines of code. As the implementation is intended to be self-explanatory and intuitively correct, several optimisations have been omitted. However, as the normalisation can easily normalise formulae consisting of several thousand $\mathcal{H}_1$ -clauses, it suffices for our needs, and the simplicity of the code leaves less room for errors.

Applying the iterative scheme to the Yahalom analysis problem presented above, i.e. the conjunction of Table 5 and Table 6, yields the following specialised clauses, after a single iteration of the algorithm:

1.	$\text{net}(\langle a, n_a \rangle)$
2.	$\text{net}(\langle b, \{ \langle X_a, X_{n_a}, n_b \rangle \}_{k_{bs}} \rangle) \Leftarrow \text{net}(\langle X_a, X_{n_a} \rangle)$
3.	$\text{net}(\langle \{ \langle b, k_{ab}, X_{n_a}, n_b \rangle \}_{k_{as}}, \{ \langle a, k_{ab} \rangle \}_{k_{bs}} \rangle) \Leftarrow \text{net}(\langle b, \{ \langle a, X_{n_a}, n_b \rangle \}_{k_{bs}} \rangle)$
4.	$\text{net}(\langle \{ \langle a, k_{ab} \rangle \}_{k_{bs}}, \{ n_b \}_{k_{ab}} \rangle, \text{a_key}(k_{ab})) \Leftarrow \text{net}(\langle \{ \langle b, k_{ab}, n_a, n_b \rangle \}_{k_{as}}, \{ \langle a, k_{ab} \rangle \}_{k_{bs}} \rangle)$
5.	$\text{b_key}(k_{ab}) \Leftarrow \text{net}(\langle \{ \langle a, k_{ab} \rangle \}_{k_{bs}}, \{ n_b \}_{k_{ab}} \rangle)$

We omit the clauses for the attacker, as all variables occurring in them have trivially infinite substitution sets, thus these clauses remain unchanged.

³ available at <http://www.imm.dtu.dk/~{}crn>

Observe first, that the formula is now contained in $\mathcal{H}_1$, and thus the resulting normal form represents its least model. This means that the analysis result is complete, and inspecting it shows that `a_key` and `b_key` can only be bound to k_{ab} . Hence one instance of the protocol serves its purpose and will always successfully establish the intended shared key between the principals. We also find that $k_{as}, k_{bs}, k_{ab} \notin \text{dy}$, which guarantees confidentiality of all the shared keys, including the newly distributed one. Both of these results were expected as Yahalom is, as already mentioned, considered a secure protocol.

6 Conclusion

Horn clauses have many applications. One example, as we have shown, is that it allows for a clear and intuitive specification of protocol analysis problems. However, as is often the case with clean and simple analysis specifications, the resulting instances of analysis problems are outside any of the known decidable fragments of Horn clauses, threatening the analysis itself with being a mere theoretical exercise.

This paper presents an iterative scheme for solving Horn clauses that guarantees termination and soundness, and may in some cases even give completeness. It relies on a chosen known decidable class of Horn clauses and a sound approximation method; the latter being achieved by iteratively specialising the analysis problem and improving the analysis result, until either no further improvements can be made or the analysis result is complete.

We also sketched a methodology for specifying static analyses of cryptographic protocols, in particular we validated the non-trivial key-distribution protocol Yahalom. Our example was concerned with only a single instance of the protocol, but even this simple scenario was sufficient to push the problem beyond the known decidable classes. This demonstrates the complications of containing problems within the decidable domains of Horn logic, but also illustrates the usability of the scheme, as it still successfully determined a complete result for the analysis problem.

We expect the iterative scheme to also prove useful beyond protocol analysis, as deciding the least model or a sufficiently small model of Horn clause-based problems, is attractive to many different branches of theoretical computer science.

References

1. Tärnlund, S.Å.: Horn clause computability. *BIT* 17(2), 215–226 (1977)
2. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. *J. ACM* 23(4), 733–742 (1976)
3. Nielson, F., Nielson, H.R., Seidl, H.: Normalizable horn clauses, strongly recognizable relations, and spi. In: Hermenegildo, M.V., Puebla, G. (eds.) *SAS 2002*. LNCS, vol. 2477, pp. 20–35. Springer, Heidelberg (2002)
4. Goubault-Larrecq, J., Parrennes, F.: Cryptographic protocol analysis on real c code. In: Cousot, R. (ed.) *VMCAI 2005*. LNCS, vol. 3385, pp. 363–379. Springer, Heidelberg (2005)

5. Goubault-Larrecq, J.: Deciding h_1 by resolution. *Inf. Process. Lett.* 95(3), 401–408 (2005)
6. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. *ACM Trans. Comput. Syst.* 8(1), 18–36 (1990)
7. Paulson, L.C.: Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security* 9(3), 197–216 (2001)
8. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. *Journal of Computer Security* 13(3), 347–390 (2005)
9. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: CSFW, pp. 82–96. IEEE Computer Society, Los Alamitos (2001)
10. Seidl, H., Verma, K.N.: Cryptographic protocol verification using tractable classes of horn clauses. In: Reps, T., Sagiv, M., Bauer, J. (eds.) *Wilhelm Festschrift. LNCS*, vol. 4444, pp. 97–119. Springer, Heidelberg (2007)
11. Dolev, D., Yao, A.C.C.: On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2), 198–207 (1983)
12. Nielson, F., Nielson, H.R., Hansen, R.R.: Validating firewalls using flow logics. *Theor. Comput. Sci.* 283(2), 381–418 (2002)