



Division of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

**Finding Counterexamples to Inductive Conjectures and Discovering
Security Protocol Attacks**

by

Graham Steel, Alan Bundy, Ewen Denney

Informatics Research Report EDI-INF-RR-0141

Division of Informatics
<http://www.informatics.ed.ac.uk/>

July 2002

Finding Counterexamples to Inductive Conjectures and Discovering Security Protocol Attacks

Graham Steel, Alan Bundy, Ewen Denney

Informatics Research Report EDI-INF-RR-0141

DIVISION *of* INFORMATICS

Centre for Intelligent Systems and their Applications

July 2002

appears in Proceedings of 2002 Workshop on Foundations of Computer Security

Abstract :

We present an implementation of a method for finding counterexamples to universally quantified conjectures in first-order logic. Our method uses the proof by consistency strategy to guide a search for a counterexample and a standard first-order theorem prover to perform a concurrent check for inconsistency. We explain briefly the theory behind the method, describe our implementation, and evaluate results achieved on a variety of incorrect conjectures from various sources.

Some work in progress is also presented: we are applying the method to the verification of cryptographic security protocols. In this context, a counterexample to a security property can indicate an attack on the protocol, and our method extracts the trace of messages exchanged in order to effect this attack. This application demonstrates the advantages of the method, in that quite complex side conditions decide whether a particular sequence of messages is possible. Using a theorem prover provides a natural way of dealing with this. Some early results are presented and we discuss future work.

Keywords : security protocols, cryptography, verification, counterexamples, proof by consistency

Copyright © 2002 by The University of Edinburgh. All Rights Reserved

The authors and the University of Edinburgh retain the right to reproduce and publish this paper for non-commercial purposes.

Permission is granted for this report to be reproduced by others for non-commercial purposes as long as this copyright notice is reprinted in full in any reproduction. Applications to make other use of the material should be addressed in the first instance to Copyright Permissions, Division of Informatics, The University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland.

Finding Counterexamples to Inductive Conjectures and Discovering Security Protocol Attacks

Graham Steel, Alan Bundy, and Ewen Denney
Division of Informatics
University of Edinburgh

{grahams, bundy, ewd}@dai.ed.ac.uk

Abstract

We present an implementation of a method for finding counterexamples to universally quantified conjectures in first-order logic. Our method uses the proof by consistency strategy to guide a search for a counterexample and a standard first-order theorem prover to perform a concurrent check for inconsistency. We explain briefly the theory behind the method, describe our implementation, and evaluate results achieved on a variety of incorrect conjectures from various sources.

Some work in progress is also presented: we are applying the method to the verification of cryptographic security protocols. In this context, a counterexample to a security property can indicate an attack on the protocol, and our method extracts the trace of messages exchanged in order to effect this attack. This application demonstrates the advantages of the method, in that quite complex side conditions decide whether a particular sequence of messages is possible. Using a theorem prover provides a natural way of dealing with this. Some early results are presented and we discuss future work.

1 Introduction

Inductive theorem provers are frequently employed in the verification of programs, algorithms and protocols. However, programs and algorithms often contain bugs, and protocols may be flawed, causing the proof attempt to fail. It can be hard to interpret a failed proof attempt: it may be that some additional lemmas need to be proved or a generalisation made. In this situation, a tool which can not only detect an incorrect conjecture, but also supply a counterexample in order to allow the user to identify the bug or flaw, is potentially very valuable. The problem of cryptographic security protocol verification is a specific area in which incorrect conjectures are of great consequence. If a security conjecture turns out to be false, this can indicate an attack on the protocol. A counterexample can help the user to see how the protocol can be attacked. Incorrect conjectures

also arise in automatic inductive theorem provers where generalisations are speculated by the system. Often we encounter the problem of over-generalisation: the speculated formula is not a theorem. A method for detecting these over-generalisations is required.

Proof by consistency is a technique for automating inductive proofs in first-order logic. Originally developed to prove correct theorems, this technique has the property of being refutation complete, i.e. it is able to refute in finite time conjectures which are inconsistent with the set of hypotheses. When originally proposed, this technique was of limited applicability. Recently, Comon and Nieuwenhuis have drawn together and extended previous research to show how it may be more generally applied, [10]. They describe an experimental implementation of the inductive completion part of the system. However, the check for refutation or consistency was not implemented. This check is necessary in order to ensure a theorem is correct, and to automatically refute an incorrect conjecture. We have implemented a novel system integrating Comon and Nieuwenhuis' experimental prover with a concurrent check for inconsistency. By carrying out the check in parallel, we are able to refute incorrect conjectures in cases where the inductive completion process fails to terminate. The parallel processes communicate via sockets using Linda, [8].

The ability of the technique to prove complex inductive theorems is as yet unproven. That does not concern us here – we are concerned to show that it provides an efficient and effective method for refuting *incorrect* conjectures. However, the ability to prove at least many small theorems helps alleviate a problem reported in Protzen's work on disproving conjectures, [22] – that the system terminates only at its depth limit in the case of a small unsatisfiable formula, leaving the user or proving system none the wiser.

We have some early results from our work in progress, which is to apply the technique to the aforementioned problem of cryptographic security protocol verification. These protocols often have subtle flaws in them that are not detected for years after they have been proposed. By

devising a first-order version of Paulson’s inductive formalism for the protocol verification problem, [21], and applying our refutation system, we can not only detect flaws but also automatically generate the sequence of messages needed to expose these flaws. By using an inductive model with arbitrary numbers of agents and runs rather than the finite models used in most model-checking methods, we have the potential to synthesise parallel session and replay attacks where a single principal may be required to play multiple roles in the exchange.

In the rest of the paper, we first review the literature related to the refutation of incorrect conjectures and proof by consistency, then we briefly examine the Comon-Nieuwenhuis method. We describe the operation of the system, relating it to the theory, and present and evaluate the results obtained so far. The system has been tested on a number of examples from various sources including Protzen’s work [22], Reif et al.’s, [24], and some of our own. Our work in progress on the application of the system to the cryptographic protocol problem is then presented. Finally, we describe some possible further work and draw some conclusions.

2 Literature Review

2.1 Refuting Incorrect Conjectures

At the CADE-15 workshop on proof by mathematical induction, it was agreed that the community should address the issue of dealing with non-theorems as well as theorems¹. However, relatively little work on the problem has since appeared. In the early nineties Protzen presented a sound and complete calculus for the refutation of faulty conjectures in theories with free constructors and complete recursive definitions, [22]. The search for the counterexample is guided by the recursive definitions of the function symbols in the conjecture. A depth limit ensures termination when no counterexample can be found.

More recently, Reif et al., [25], have implemented a method for counterexample construction that is integrated with the interactive theorem prover KIV, [23]. Their method incrementally instantiates a formula with constructor terms and evaluates the formulae produced using the simplifier rules made available to the system during proof attempts. A heuristic strategy guides the search through the resulting subgoals for one that can be reduced to *false*. If such a subgoal is not found, the search terminates when all variables of generated sorts have been instantiated to constructor terms. In this case the user is left with a model condition, which must be used to decide whether the instantiation found is a valid counterexample.

¹The minutes of the discussion are available from <http://www.cee.hw.ac.uk/~air/cade15/cade-15-mind-ws-session-3.html>.

Ahrendt has proposed a refutation method using model construction techniques, [1]. This is restricted to free datatypes, and involves the construction of a set of suitable clauses to send to a model generation prover. As first reported, the approach was not able in general to find a refutation in finite time, but new work aims to address this problem, [2].

2.2 Proof by Consistency

Proof by consistency is a technique for automating inductive proof. It has also been called *inductionless induction*, and *implicit induction*, as the actual induction rule used is described implicitly inside a proof of the conjecture’s consistency with the set of hypotheses. Recent versions of the technique have been shown to be *refutation complete*, i.e. are guaranteed to detect non-theorems in finite time.² The proof by consistency technique was developed to solve problems in equational theories, involving a set of equations defining the *initial model*³, E . The first version of the technique was proposed by Musser, [20], for equational theories with a *completely defined equality predicate*. This requirement placed a strong restriction on the applicability of the method. The completion process used to deduce consistency was the Knuth-Bendix algorithm, [17].

Huet and Hullot [14] extended the method to theories with free constructors, and Jouannaud and Kounalis, [15], extended it further, requiring that E should be a convergent rewrite system. Bachmair, [4], proposed the first refutationally complete deduction system for the problem, using a *linear strategy* for inductive completion. This is a restriction of the Knuth-Bendix algorithm which entails only examining overlaps between axioms and conjectures. The key advantage of the restricted completion procedure was its ability to cope with unoriented equations. The refutationally completeness of the procedure was a direct result of this.

The technique has been extended to the non-equational case. Ganzinger and Stuber, [12], proposed a method for proving consistency for a set of first-order clauses with equality using a refutation complete linear system. Kounalis and Rusinowitch, [18], proposed an extension to conditional theories, laying the foundations for the method implemented in the SPIKE theorem, [6]. Ideas from the proof by consistency technique have been used in other induction methods, such as cover set induction, [13], and test set induction, [5].

²Such a technique must necessarily be incomplete with respect to proving theorems correct, by Gödel’s incompleteness theorem.

³The initial or standard model is the minimal Herbrand model. This is unique in the case of a purely equational specification.

2.3 Cryptographic Security Protocols

Cryptographic protocols are used in distributed systems to allow agents to communicate securely. Assumed to be present in the system is a spy, who can see all the traffic in the network and may send malicious messages in order to try and impersonate users and gain access to secrets. Clark and Jacob’s survey, [9], and Anderson and Needham’s article, [3], are good introductions to the field.

Although security protocols are usually quite short, typically 2–5 messages, they often have subtle flaws in them that may not be discovered for many years. Researchers have applied various formal methods techniques to the problem, to try to find attacks on faulty protocols, and to prove correct protocols secure. These approaches include belief logics such as the so-called BAN logic, [7], state-machines, [11, 16], model-checking, [19], and inductive theorem proving, [21]. Each approach has its advantages and disadvantages. For example, the BAN logic is attractively simple, and has found some protocol flaws, but has missed others. The model checking approach can find flaws very quickly, but can only be applied to finite (and typically very small) instances of the protocol. This means that if no attack is found, there may still be an attack upon a larger instance. Modern state machine approaches can also find and exhibit attacks quickly, but require the user to choose and prove lemmas in order to reduce the problem to a tractable finite search space. The inductive method deals directly with the infinite state problem, and assumes an arbitrary number of protocol participants, but proofs are tricky and require days or weeks of expert effort. If a proof breaks down, there are no automated facilities for the detection of an attack.

3 The Comon-Nieuwenhuis Method

Comon and Nieuwenhuis, [10], have shown that the previous techniques for proof by consistency can be generalised to the production of a first-order axiomatisation \mathcal{A} of the minimal Herbrand model such that $\mathcal{A} \cup E \cup C$ is consistent if and only if C is an inductive consequence of E . With \mathcal{A} satisfying the properties they define as an *I-Axiomatisation*, inductive proofs can be reduced to first-order consistency problems and so can be solved by any saturation based theorem prover. We give a very brief summary of their results here. Suppose I is, in the case of Horn or equational theories, the unique minimal Herbrand model, or in the case of non-Horn theories, the so-called *perfect model* with respect to a total ordering on terms, \succ^4 :

Definition 1 *A set of first-order formulae \mathcal{A} is an I-Axiomatisation of I if*

⁴Saturation style theorem proving always requires that we have such an ordering available.

1. \mathcal{A} is a set of purely universally quantified formulae
2. I is the only Herbrand model of $E \cup \mathcal{A}$ up to isomorphism.

An I-Axiomatisation is normal if $\mathcal{A} \models s \neq t$ for all pairs of distinct normal terms s and t

The I-Axiomatisation approach produces a clean separation between the parts of the system concerned with inductive completion and inconsistency detection. Completion is carried out by a saturation based theorem prover, with inference steps restricted to those produced by conjecture superposition, a restriction of the standard superposition rule. Only overlaps between conjecture clauses and axioms are considered. Each non-redundant clause derived is checked for consistency against the I-Axiomatisation. If the theorem prover terminates with saturation, the set of formulae produced comprise a *fair induction derivation*. The key result of the theory is this:

Theorem 1 *Let \mathcal{A} be a normal I-Axiomatisation, and C_0, C_1, \dots be a fair induction derivation. Then $I \models C_0$ iff $\mathcal{A} \cup \{c\}$ is consistent for all clauses c in $\bigcup_i C_i$.*

This theorem is proved in [10]. Comon and Nieuwenhuis have shown that this conception of proof by consistency generalises and extends earlier approaches. An equality predicate as defined by Musser, a set of free constructors as proposed by Huet and Hullot or a ground reducibility predicate as defined by Jouannaud and Kounalis could all be used to form a suitable I-Axiomatisation. The technique is also extended beyond ground convergent specifications (equivalent to saturated specifications for first-order clauses) as required in [15, 4, 12]. Previous methods, e.g. [6], have relaxed this condition by using conditional equations. However a ground convergent rewrite system was still required for deducing inconsistency. Using the I-Axiomatisation method, conjectures can be proved or refuted in (possibly non-free) constructor theories which cannot be specified by a convergent rewrite system.

Whether these extensions to the theory allow larger theorems to be *proved* remains to be seen, and is not of interest to us here. We are interested in how the wider applicability of the method can allow us to investigate the ability of the proof by consistency technique to root out a counterexample to realistic *incorrect* conjectures.

4 Implementation

Figure 1 illustrates the operation of our system. The input is an inductive problem in Saturate format and a normal I-Axiomatisation (see Definition 1, above). The version of Saturate customised by Nieuwenhuis for implicit induction (the right hand box in the diagram) gets the problem file only, and proceeds to pursue inductive completion, i.e. to derive a fair induction derivation. Every

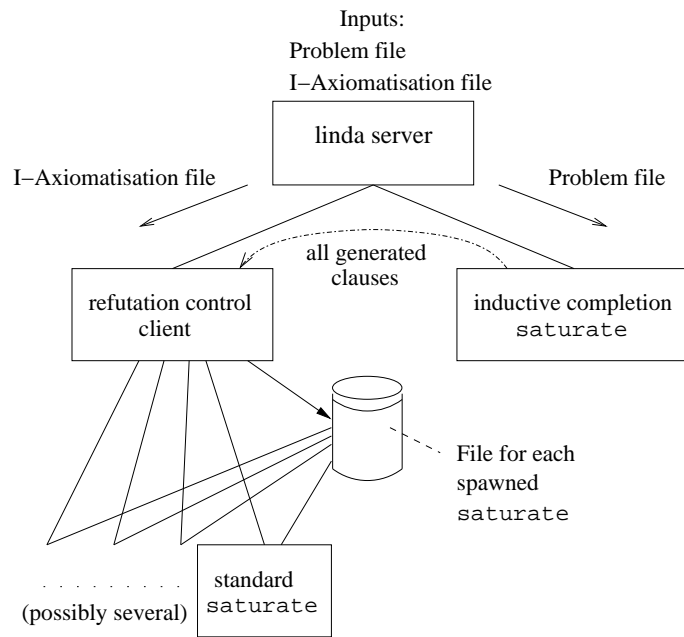


Figure 1: System operation

non-redundant clause generated is passed via the server to the refutation control program (the leftmost box). For every new clause received, this program generates a problem file containing the I-Axiomatisation and the new clause, and spawns a standard version of Saturate to check the consistency of the file. Crucially, these spawned Saturates are not given the original axioms – only the I-Axioms are required, by Theorem 1. This means that almost all of the search for an inconsistency is done by the prover designed for inductive problems and the spawned Saturates are just used to check for inconsistencies between the new clauses and the I-Axiomatisation. This should lead to a false conjecture being refuted after fewer inference steps have been attempted than if the conjecture had been given to a standard first-order prover together with all the axioms and I-Axioms. We evaluate this in the next section.

If, at any time, a refutation is found by a spawned prover, the proof is written to a file and the completion process and all the other spawned Saturate processes are killed. If completion is reached by the induction prover, this is communicated to the refutation control program, which will then wait for the results from the spawned processes. If they all terminate with saturation, then there are no inconsistencies, and so the theorem has been proved (by Theorem 1).

There are several advantages to the parallel architecture we have employed. One is that it allows us to refute incorrect conjectures even if the inductive completion process does not terminate. This would also be possible by modifying the main induction Saturate to check each

clause in turn, but this would result in a rather messy and unwieldy program. Another advantage is that we are able to easily devote a machine solely to inductive completion in the case of harder problems. It is also very convenient when testing a new model to be able to just look at the deduction process before adding the consistency check later on, and we preserve the attractive separation in the theory between the deduction and the consistency checking processes.

A disadvantage of our implementation is that launching a new Saturate process to check each clause against the I-Axiomatisation generates some overheads in terms of disk access etc. In our next implementation, when a spawned prover reaches saturation (i.e. no inconsistencies), it will clear its database and ask the refutation control client for another clause to check, using the existing sockets mechanism. This will cut down the amount of memory and disk access required. A further way to reduce the consistency checking burden is to take advantage of knowledge about the structure of the I-Axiomatisation for simple cases. For example, in the case of a free constructor specification, the I-Axiomatisation will consist of clauses specifying the inequality of non-identical constructor terms. Since it will include no rules referring to defined symbols, it is sufficient to limit the consistency check to generated clauses containing only constructors and variables.

Table 1: Sample of results. In the third column, the first number shows the number of clauses derived by the inductive completion prover, and the number in brackets indicates the number of clauses derived by the parallel checker to spot the inconsistency. The fourth column shows the number of clauses derived by an unmodified first-order prover when given the conjecture, axioms and I-Axioms all together.

Problem	Counterexample found	No. of clauses derived to find refutation	No. of clauses derived by a standard prover
$\forall N, M. \neg(s(N) + M = s(0))$	$N = 0, M = 0$	2(+0)	2
$X \neq Y \wedge X \neq 0 \wedge Y \neq 0$ $\Rightarrow (X \geq Y \wedge Y \neq 0) \vee$ $(X \neq 0 \wedge Y = 0)$	$X = s(0),$ $Y = s(s(X))$	4 (+3)	6
$app(K, L) = app(L, K)$	$K = 0, L = s(X)$	9(+11)	stuck in loop
$sort(l_1) \wedge l_2 = ap(l_1, [head(l_3)])$ $\wedge length(l_3) \geq 2 * length(l_1)$ $\wedge l_3 \neq nil \wedge$ $member(head(l_1), tail(l_3))$ $\Rightarrow sort(l_2)$	$l_1 = [s(X),$ $l_2 = [s(X), 0]$ $l_3 = [0, s(X) Y]$	55(+1)	76
All graphs are acyclic	$[e(a, a)]$	99	123
All loopless graphs are acyclic	$[e(s(a), a), e(a, s(a))]$	178	2577
$gcd(X, X) = 0$	$X = s(0)$	17(+2)	29
Impossibility property for Neuman-Stubblefield key exchange protocol	$[msg(1), msg(2),$ $msg(3), msg(4)]$	866 (+0)	1733
Authenticity property for simple protocol from [9]	see section 6	730(+1)	3148

5 Evaluation of Results

Table 1 shows a sample of results achieved so far. The first three examples are from Protzen’s work, [22], the next two from Reif et al.’s, [25], and the last three are from our work. The *gcd* example is included because previous methods of proof by consistency could not refute this conjecture. Comon and Nieuwenhuis showed how it could be tackled, [10], and here we confirm that their method works. The last two conjectures are about properties of security protocols. The ‘impossibility property’ states that no trace reaches the end of a protocol. Its refutation comprises the proof of a possibility property, which is the first thing proved about a newly modelled protocol in Paulson’s method, [21]. The last result is the refutation of an authenticity property, indicating an attack on the protocol. This protocol is a simple example included in Clark’s survey, [9], for didactic purposes, but requires that one principal play both roles in a protocol run. More details are given in section 6.

Our results on Reif et al.’s examples do not require the user to verify a model condition, as the system described in their work does. Interestingly, the formula remaining as a model condition in their runs is often the same as the formula which gives rise to the inconsistency when checked against the I-Axiomatisation in our runs. This is because the KIV system stops when it derives a term

containing just constructors and variables. In such a case, our I-Axiomatisation would consist of formulae designed to check validity of these terms. This suggests a way to automate the model condition check in the KIV system.

On comparing the number of clauses derived by our system and the number of clauses required by a standard first-order prover (SPASS), we can see that the proof by consistency strategy does indeed cut down on the number of inferences required. This is more evident in the larger examples. Also, the linear strategy allows us to cope with commutativity conjectures, like the third example, which cause a standard prover to go into a loop. We might ask: what elements of the proof by consistency technique are allowing us to make this saving in required inferences? One is the refutation completeness result for the linear strategy, so we know we need only consider overlaps between conjectures and axioms. Additionally, separating the I-Axioms from the theory axioms reduces the number of overlaps between conjectures and axioms to be considered each time. We also use the results about inductively complete positions for theories with free constructors, [10]. This applies to all the examples except those in graph theory, where we used Reif’s formalism and hence did not have free constructors. This is the probable reason why, on these two examples, our system did not make as large a saving in derived clauses.

The restriction to overlaps between conjectures and ax-

ioms is similar in nature to the so-called set of support strategy, using the conjecture as the initial supporting set. The restriction in our method is tighter, since we don't consider overlaps between formulae in the set of support. Using the set of support strategy with the standard prover on the examples in Table 1, refutations are found after deriving fewer clause than required by the standard strategy. However, performance is still not as good as for our system, particularly in the free constructor cases. The set of support also doesn't fix the problem of divergence on un-oriented conjectures, like the commutativity example.

The efficiency of the method in terms of clauses derived compared to a standard prover looks good. However, actual time taken by our system is much longer than that for the standard SPASS. This is because the Saturate prover is rather old, and was not designed to be a serious tool for large scale proving. In particular, it does not utilise any term indexing techniques, and so redundancy checks are extremely slow. As an example, the impossibility property took about 50 minutes to refute in Saturate, but about 40 seconds in SPASS, even though more than twice as many clauses had to be derived. We used Saturate in our first system as Nieuwenhuis had already implemented the proof by consistency strategy in the prover. A re-implementation of the whole system using SPASS should give us even faster refutations, and is one of our next tasks.

Finally, we also tested the system on a number of small inductive theorems. Being able to prove small theorems allows us to attack a problem highlighted in Protzen's work: that if an candidate generalisation (say) is given to the counterexample finder and it returns a result saying that the depth limit was reached before a counterexample was found, the system is none the wiser as to whether the generalisation is worth pursuing. If we are able to prove at least small examples to be theorems, this will help alleviate the problem. Our results were generally good: 7 out of 8 examples we tried were proved, but one was missed. Comon intends to investigate the ability of the technique to prove more and larger theorems in future.

More details of the results including some sample runs and details of the small theorems proved can be found at <http://www.dai.ed.ac.uk/~grahams/linda>.

6 Application to Cryptographic Security Protocols

We now describe some work in progress on applying our technique to the cryptographic security protocol problem. As we saw in section 2.3, one of the main thrusts of research has been to apply formal methods to the problem. Researchers have applied techniques from model checking, theorem proving and modal logics amongst others. Much attention is paid to the modelling of the abilities

of the spy in these models. However, an additional consideration is the abilities of the participants. Techniques assuming a finite model, with typically two agents playing distinct roles, often rule out the possibility of discovering a certain kind of parallel session attack, in which one participant plays both roles in the protocol. The use of an inductive model allows us to discover these kind of attacks. An inductive model also allows us to consider protocols with more than two participants, e.g. conference-key protocols.

Paulson's inductive approach has been used to verify properties of several protocols, [21]. Protocols are formalised in typed higher-order logic as the set of all possible traces, a trace being a list of events like 'A sends message X to B '. This formalism is mechanised in the Isabelle/HOL interactive theorem prover. Properties of the security protocol can be proved by induction on traces. The model assumes an arbitrary number of agents, and any agent may take part in any number of concurrent protocol runs playing any role. Using this method, Paulson discovered a flaw in the simplified Otway-Rees shared key protocol, [7], giving rise to a parallel session attack where a single participant plays both protocol roles. However, as Paulson observed, a failed proof state can be difficult to interpret in these circumstances. Even an expert user will be unsure as to whether it is the proof attempt or the conjecture which is at fault. By applying our counterexample finder to these problems, we can automatically detect and present attacks when they exist.

Paulson's formalism is in higher-order logic. However, no 'fundamentally' higher-order concepts are used – in particular there is no unification of higher-order objects. Objects have types, and sets and lists are used. All this can be modelled in first-order logic. The security protocol problem has been modelled in first-order logic before, e.g. by Weidenbach, [26]. This model assumed a two agent model with just one available nonce⁵ and key, and so could not detect the kind of parallel session attacks described. Our model allows an arbitrary number of agents to participate, playing either role, and using an arbitrary number of fresh nonces and keys.

6.1 Our Protocol Model

Our models aims to be as close as possible to a first-order version of Paulson's formalism. As in Paulson's model, agents, nonces and messages are free data types. This allows us to define a two-valued function eq which will tell us whether two pure constructor terms are equal or not. Since the rules defining eq are exhaustive, they also have the effect of suggesting instantiations where certain conditions must be met, e.g. if we require the identities of two agents to be distinct. The model is kept Horn by defining two-valued functions for checking the side conditions for a message to be sent, e.g. we define conditions for

⁵A nonce is a unique identifying number.

$member(X, L) = true$ and $member(X, L) = false$ using our *eq* function. This cuts down the branching rate of the search.

The intruder's knowledge is specified in terms of sets. Given a trace of messages exchanged, XT , we define $analz(XT)$ to be the least set including XT closed under projection and decryption by known keys. This is accomplished by using exactly the same rules as the Paulson model, [21, p. 12]. Then, we can define the messages the intruder may send, given a trace XT , as being members of the set $synth(analz(XT))$, where $synth(X)$ is the least set including agent names closed under pairing and encryption by known keys. Again this set is defined in our model with the same axioms that Paulson uses.

A trace of messages is modelled as a list. For a specific protocol, we generally require one axiom for each protocol message. These axioms take the form of rules with the informal interpretation, 'if XT is a trace containing message n addressed to agent xa , then the trace may be extended by xa responding with message $n + 1$ '. Once again, this is very similar to the Paulson model.

An example illustrates some of these ideas. In Figure 2 we demonstrate the formalism of a very simple protocol included in Clark and Jacob's survey to demonstrate parallel session attacks, [9]. Although simple, the attack on the protocol does require principal A to play the role of both initiator and responder. It assumes that A and B already share a secure key, K_{AB} . N_A denotes a nonce generated by A .

In a symmetric key protocol, principals should respond to $key(A, B)$ and $key(B, A)$, as they are in reality the same. At the moment we model this with two possible rules for message 2, but it should be straightforward to extend the model to give a cleaner treatment of symmetric keys as sets of agents. Notice we allow a principal to respond many times to the same message, as Paulson's formalism does.

The second box, Figure 3, shows how the refutation of a conjectured security property leads to the discovery of the known attack. At the moment, choosing which conjectures to attempt to prove is tricky. A little thought is required in order to ensure that only a genuine attack can refute the conjecture. More details of our model for the problem, including the specification of intruder knowledge, can be found at <http://www.dai.ed.ac.uk/~grahams/linda>.

This application highlights a strength of our refutation system: in order to produce a backwards style proof, as Paulson's system does, we must apply rules with side conditions referring as yet uninstantiated variables. For example, a rule might be applied with the informal interpretation, 'if the spy can extract X from the trace of messages sent up to this point, then he can break the security conjecture'. At the time the rule is applied, X will be uninstantiated. Further rules instantiate parts of the trace,

and side conditions are either satisfied and eliminated, or found to be unsatisfiable, causing the clauses containing the condition to be pruned off as redundant. The side conditions influence the path taken through the search space, as smaller formulae are preferred by the default heuristic in the prover. This means that some traces a naïve counterexample search might find are not so attractive to our system, e.g. a trace which starts with several principals sending message 1 to other principals. This will not be pursued at first, as all the unsatisfied side conditions will make this formula larger than others.

7 Further Work

Our first priority is to re-implement the system using SPASS, and then to carry out further experiments with larger false conjectures and more complex security protocols. This will allow us to evaluate the technique more thoroughly. A first goal is to rediscover the parallel session attack discovered by Paulson. The system should also be able to discover more standard attacks, and the Clark survey, [9], provides a good set of examples for testing. We will then try the system on other protocols and look for some new attacks. A key advantage of our security model is that it allows attacks involving arbitrary numbers of participants. This should allow us to investigate the security of protocols involving many participants in a single run, e.g. conference key protocols.

In future, we also intend to implement more sophisticated heuristics to improve the search performance, utilising domain knowledge about the security protocol problem. Heuristics could include eager checks for unsatisfiable side conditions. Formulae containing these conditions could be discarded as redundant. Another idea is to vary the weight ascribed to variables and function symbols, so as to make the system inclined to check formulae with predominantly ground variables before trying ones with many uninstantiated variables. This should make paths to attacks more attractive to the search mechanism, but some careful experimentation is required to confirm this.

The Comon-Nieuwenhuis technique has some remaining restrictions on applicability, in particular the need for *reductive definitions*, a more relaxed notion of reducibility than is required for ground convergent rewrite systems. It is quite a natural requirement that recursive function definitions should be reducing in some sense. For example, the model of the security protocol problem is reductive in the sense required by Comon and Nieuwenhuis. Even so, it should be possible to extend the technique for non-theorem detection in the case of non-reductive definitions, at the price of losing any reasonable chance of proving a theorem, but maintaining the search guidance given by the proof by consistency technique. This would involve allowing inferences by standard superposition if conjecture superposition is not applicable.

The Clark-Jacob protocol demonstrating parallel session attacks. At the end of a run, A should now be assured of B 's presence, and has accepted nonce N_A to identify authenticated messages.

1. $A \rightarrow B : \{ \! \! \! \{ N_A \} \! \! \! \}_{K_{AB}}$
2. $B \rightarrow A : \{ \! \! \! \{ N_A + 1 \} \! \! \! \}_{K_{AB}}$

Formula for modelling message 1 of the protocol. Informally: if XT is a trace, XA and XB agents, and XNA a number not appearing as a nonce in a previous run, then the trace may be extended by XA initiating a run, sending message 1 of the protocol to XB .

$$\begin{aligned} m(XT) = true \wedge agent(XA) = true \wedge agent(XB) = true \wedge number(XNA) = true \\ \wedge member(sent(X, Y, encr(nonce(XNA), K)), XT) = false \Rightarrow \\ m([sent(XA, XB, encr(nonce(XNA), key(XA, XB)))]|XT) = true \end{aligned}$$

Formulae for message 2. Two formulae are used to make the response to the shared key symmetric (see text). Informally: if XT is a trace containing message 1 of the protocol addressed to agent XB , encrypted under a key he shares with agent XA , then the trace may be extended by agent XB responding with message 2.

$$member(sent(X, XB, encr(nonce(XNA), key(XA, XB))), XT) = true \wedge m(XT) = true \Rightarrow \\ m([sent(XB, XA, encr(s(nonce(XNA)), key(XA, XB)))]|XT) = true.$$

$$member(sent(X, XB, encr(nonce(XNA), key(XB, XA))), XT) = true \wedge m(XT) = true \Rightarrow \\ m([sent(XB, XA, encr(s(nonce(XNA)), key(XB, XA)))]|XT) = true.$$

Figure 2: The modelling of the Clark-Jacob protocol

The parallel session attack suggested by Clark and Jacob [9]. At the end of the attack, A believes B is operational. B may be absent or may no longer exist:

1. $A \rightarrow C_B : \{ \! \! \! \{ N_A \} \! \! \! \}_{K_{AB}}$
2. $C_B \rightarrow A : \{ \! \! \! \{ N_A \} \! \! \! \}_{K_{AB}}$
3. $A \rightarrow C_B : \{ \! \! \! \{ s(N_A) \} \! \! \! \}_{K_{AB}}$
4. $C_B \rightarrow A : \{ \! \! \! \{ s(N_A) \} \! \! \! \}_{K_{AB}}$

Below is the incorrect security conjecture, the refutation of which gives rise to the attack above. Informally this says, 'for all valid traces T , if A starts a run with B using nonce N_A , and receives the reply $s(N_A)$ from principal X , and no other principal has sent a reply, then the reply must have come from agent B .'

$$\begin{aligned} member(sent(XA, XB, encr(nonce(XNA), K)), XT) = true \\ \wedge XT = [sent(X, XA, encr(s(nonce(XNA)), K))|T] \\ \wedge member(sent(Y, XA, encr(s(nonce(XNA)), K)), T) = false \\ \wedge m(XT) = true \Rightarrow eq(X, XB) = true \end{aligned}$$

The final line of output from the system, giving the attack.

```
c(sent(spy, a, encr(s(nonce(0)), key(a, s(a))))) ,
c(sent(a, s(a), encr(s(nonce(0)), key(a, s(a))))) ,
c(sent(spy, a, encr(nonce(0), key(a, s(a))))) ,
c(sent(a, s(a), encr(nonce(0), key(a, s(a))))) , nil))))
```

Figure 3: The attack and its discovery

8 Conclusions

In this paper we have presented a working implementation of a novel method for investigating an inductive conjecture, with a view to proving it correct or refuting it as false. We are primarily concerned with the ability of the system to refute false conjectures, and have shown results from testing on a variety of examples. These have shown that our parallel inductive completion and consistency checking system requires considerably fewer clauses to be derived than a standard first-order prover does when tackling the whole problem at once. The application of the technique to producing attacks on faulty cryptographic security protocols looks promising, and the system has already synthesised an attack of a type many finite security models will not detect. We intend to produce a faster implementation using the SPASS theorem prover, and then to pursue this application further.

References

- [1] W. Ahrendt. A basis for model computation in free data types. In *CADE-17, Workshop on Model Computation - Principles, Algorithms, Applications*, 2000.
- [2] W. Ahrendt. Deductive search for errors in free data type specifications using model generation. In *CADE-18*, 2002.
- [3] R. Anderson and R. Needham. *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, chapter Programming Satan's Computer, pages 426–440. Springer, 1995.
- [4] L. Bachmair. *Canonical Equational Proofs*. Birkhauser, 1991.
- [5] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. Rapport de Recherche 1663, INRIA, April 1992.
- [6] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.
- [7] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
- [8] N. Carreiro and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
- [9] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available via <http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz>, 1997.
- [10] H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186, May/June 2000.
- [11] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions in Information Theory*, 2(29):198–208, March 1983.
- [12] H. Ganzinger and J. Stuber. *Informatik — Festschrift zum 60. Geburtstag von Günter Hotz*, chapter Inductive theorem proving by consistency for first-order clauses, pages 441–462. Teubner Verlag, 1992.
- [13] M. S. Krishnamoorthy H. Zhang, D. Kapur. A mechanizable induction principle for equational specifications. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 1988.
- [14] G. Huet and J. Hullot. Proofs by induction in equational theories with constructors. *Journal of the Association for Computing Machinery*, 25(2), 1982.
- [15] J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. *Information and Computation*, 82(1), 1989.
- [16] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7:79–130, 1994.
- [17] D. Knuth and P. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press, 1970.
- [18] E. Kounalis and M. Rusinowitch. A mechanization of inductive reasoning. In AAAI Press and MIT Press, editors, *Proceedings of the American Association for Artificial Intelligence Conference, Boston*, pages 240–245, July 1990.
- [19] G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.
- [20] D. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM, 1980.
- [21] L.C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

- [22] M. Protzen. Disproving conjectures. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 340–354, Saratoga Springs, NY, USA, June 1992. Published as Springer Lecture Notes in Artificial Intelligence, No 607.
- [23] W. Reif. The KIV Approach to Software Verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages and Tools for the Construction of Correct Software*, volume 1009. Springer Verlag, 1995.
- [24] W. Reif, G. Schellhorn, and A. Thums. Fehlersuche in formalen Spezifikationen. Technical Report 2000-06, Fakultät für Informatik, Universität Ulm, Germany, May 2000. (In German).
- [25] W. Reif, G. Schellhorn, and A. Thums. Flaw detection in formal specifications. In *IJCAR'01*, pages 642–657, 2001.
- [26] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, LNAI 1632, pages 314–328, Trento, Italy, July 1999. Springer-Verlag.