Partial Order Reductions for Security Protocol Verification [⋆]

Edmund Clarke[1], Somesh Jha[1], and Will Marrero[1]

Department of Computer Science,
Carnegie Mellon University,
Pittsburgh, PA 15213.
{emc,sjha,marrero}@cs.cmu.edu

Abstract. In this paper we explore partial order reduction that make the task of verifying cryptographic protocols more efficient. These reduction techniques have been implemented in our tool BRUTUS. Although we have implemented several reduction techniques in our tool BRUTUS, due to space restrictions in this paper we only focus on partial order reductions. Partial order reductions have proved very useful in the domain of model checking reactive systems. These reductions are not directly applicable in our context because of additional complications caused by tracking knowledge of various agents. We present partial order reductions in the context of verifying security protocols and prove their correctness. Experimental results showing the benefits of this reduction technique are also presented.

Keywords: Model checking, partial order reductions, and security.

1 Introduction

Due to the rapid growth of such entities as "the Internet" and "the World Wide Web", computer security has recently become a very popular topic. As more and more people gain access to these shared resources, and as more services are offered, the importance of being able to provide security guarantees becomes paramount. Typically, these guarantees are provided by means of security protocols that make use of encryption. Several researchers have proposed techniques to analyze these protocols in an attempt to find errors or to prove them correct. There are three basic approaches for verifying such protocols.

One of the first attempts at formalizing the notion of a correct protocol was the Logic of Authentication, more commonly known as the BAN logic [BAN90]. This logic proved useful in analyzing security protocols. Kindred and Wing helped to automate the use of this logic by developing a theory generator for it [KW97]. However, one of the

[⋆] This research is sponsored by the National Science Foundation (NSF) under Grant No. CCR-9505472 and the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA, or the United States Government.

drawbacks of the logic is the lack of a formal model with which to define the semantics of the logic.

There has been much work recently on formal models for security protocols. A number of researchers have used general purpose model checkers to verify authentication protocols [Low97,MMS97,Ros96]. In all these cases, the users must specify the "bad traces" and check to see if any of them are valid traces of the model. In [CJM98], we describe a special purpose model checking tool for verifying authentication protocols which has a built-in adversary that can construct new messages when trying to subvert a protocol.

Bella and Paulson have used theorem proving to verify authentication protocols [BP97]. Their method requires that one express the set of all possible traces by providing a set of rules that describe how to extend a valid trace. Using the same syntax, one then describes the relationships between events that must hold true of correct traces, and Isabelle tries to prove that all valid traces are also correct traces. A theorem proving type approach is also taken by [Mea96].

Model checking based techniques for verifying security protocols suffer from the well known *state explosion problem*, i.e., the state space of the system grows exponentially in the number of components. In the domain of model checking of reactive systems there are numerous techniques for reducing the state space of the system. One such important technique is *partial order reduction*. This technique does not directly apply to our framework because we explicitly keep track of knowledge of various agents and because our logic can refer to this knowledge in a meaningful way.

Partial order reduction allows one to prune the set of traces of a system by reducing the number of inter-leavings to be considered. For example, if the system is insensitive to permuting two actions α and β, then one can consider only one interleaving (say $\alpha\beta$) and ignore the other interleaving ($\beta\alpha$) while exploring the system. This kind of reduction has proved valuable in verifying reactive systems [GPS96,Pel96,Val91]. In this paper we present partial order reduction technique as it applies to the verification of security protocols. The proof of correctness is also presented. Due to space limitations, proofs of various results are not presented, but the general structure of the proof of correctness is clearly described. The framework for our proof is fairly general so that other researchers working in this area can also use it.

The rest of this paper is organized as follows: In Section 2 we review the most common way in which messages are modelled when verifying security protocols. Sections 3 and 4 describe the computation model which we use to provide the semantics for the logic. This model is closely based on our tool, BRUTUS. The syntax and semantics of a logic capable of expressing properties of authentication and electronic commerce protocols are described in Section 5. Partial order reductions are described in Section 6. Experimental results are presented in Section 7. Related and future work are discussed in Sections 8 and 9.

2 Messages

Typically, messages exchanged during the run of a protocol are constructed from smaller sub-messages using pairing and encryption. The smallest such sub-messages (i.e.

those which contain no sub-messages themselves) are called *atomic messages*. There are four kinds of *atomic messages*.

- *Keys* are used to encrypt messages. Keys have the property that every key k has an inverse k^{-1} such that for all messages m, $\{\{m\}_k\}_{k^{-1}} = m$. (Note that for symmetric key cryptography the decryption key is the same as the encryption key, so $k = k^{-1}$.)
- *Principal names* are used to refer to the participants in a protocol.
- *Nonces* can be thought of as randomly generated numbers. The intuition is that no one can predict the value of a nonce; therefore, any message containing a nonce can be assumed to have been generated after the nonce was generated. (It is not an "old" message.)
- *Data* plays no role in how the protocol works but is intended to be communicated between the principals.

Let $\mathcal{A}$ denote the space of *atomic messages*. The set of all messages $\mathcal{M}$ over some set of atomic messages $\mathcal{A}$ is inductively defined as follows:

- If $a \in \mathcal{A}$ then $a \in \mathcal{M}$. (Any *atomic message* is a message.)
- If $m_1 \in \mathcal{M}$ and $m_2 \in \mathcal{M}$ then $m_1 \cdot m_2 \in \mathcal{M}$. (Two messages can be paired together to form a new message.)
- If $m \in \mathcal{M}$ and key $k \in \mathcal{A}$ then $\{m\}_k \in \mathcal{M}$. (A message m can be encrypted with key k to form a new message.)

We would also like to generalize the notion of messages to *message templates*. A message template can be thought of as a message containing one or more message variables. To extend messages to message templates we add the following to the inductive definition of messages:

- If v is a message variable, then $v \in \mathcal{M}$.

Since all keys have inverses, we always take advantage of the following reduction: $\{\{m\}_k\}_{k^{-1}} = m$. It is also important to note that we make the following *perfect encryption* assumption: the only way to generate $\{m\}_k$ is from m and k. In other words, for all messages m, m_1, and m_2 and keys k, $\{m\}_k \neq m_1 \cdot m_2$, and $\{m\}_k = \{m'\}_{k'} \Rightarrow m = m' \wedge k = k'$.

We also need to consider how new messages can be created from already known messages by encryption, decryption, pairing (concatenation), and projection. The following rules capture this relationship by defining how a message can be derived from some initial set of messages I.

1. If $m \in I$ then $I \vdash m$.
2. If $I \vdash m_1$ and $I \vdash m_2$ then $I \vdash m_1 \cdot m_2$. (**pairing**)
3. If $I \vdash m_1 \cdot m_2$ then $I \vdash m_1$ and $I \vdash m_2$. (**projection**)
4. If $I \vdash m$ and $I \vdash k$ for key k, then $I \vdash \{m\}_k$. (**encryption**)
5. If $I \vdash \{m\}_k$ and $I \vdash k^{-1}$ then $I \vdash m$. (**decryption**)

This defines the most common derivability relation used to model the capabilities of the adversary in the literature. Given some base set of messages I, we denote all the messages that can be derived from I as $\overline{I}$, the *closure* of I under the rules above. For example, if I_0 is some finite set of messages overheard by the adversary, then $\overline{I_0}$ represents the set of all messages known to the adversary. In general, $\overline{I}$ is infinite, but researchers have taken advantage of the fact that one need not actually compute $\overline{I}$. Once we describe the semantics of our logic, it will be clear that it suffices to check whether $m \in \overline{I}$ for some finite number of messages m. However, checking whether $m \in \overline{I}$ must still be decidable. For a detailed discussion of this question, see [CJM98].

3 The Model

We model a protocol by the asynchronous composition of a set of named communicating processes which model the honest agents and the adversary. We would like to model an insecure and lossy communication medium, in which a principal has no guarantees about the origin of a message, and where the adversary is free to eavesdrop on all communications. Therefore, in the model, we insist that all communications go through the adversary. In other words, all messages sent are intercepted by the adversary and all messages received by honest agents are actually sent by the adversary. In addition, in an attempt to subvert the protocol, the adversary is allowed to create new messages from the information it gains by eavesdropping. The adversary is also allowed to participate in the sessions as an honest agent.

In order to make the model finite, we must place a bound on the number of sessions that a principal may attempt. A session will be modelled as an instance of a principals role in the protocol. Each session is a separate copy or execution of a principal and consists of a single sequence of actions that make up that agent's role in the protocol, along with all the variable bindings and knowledge acquired during the execution [1]. An agent can have multiple sessions, but each session is executed once. When we combine these with a single session of the adversary, we get the entire model of the protocol.

Each session of an honest principal is modelled as a 5-tuple $\langle N, S, B, I, P\rangle$ where:

- $N \in$ *names* is the name of the principal.
- S is the unique *ID* for this session.
- B: *vars*$(N) \rightarrow \mathcal{M}$ is a set of bindings for *vars*(N), the set of variables appearing in principal N, which are bound for a particular session as it receives messages.
- $I \subseteq \mathcal{M}$ is the set of messages known to the principal of this session.
- P is a process description (similar in style to CSP) given as a sequence of actions to be performed. These actions include the pre-defined actions **send** and **receive**, as well as user defined internal actions such as **commit** and **debit**.

The model of the adversary, Ω, is similar to that of an honest agent or principal; however, the adversary is not bound to follow the protocol and so it does not make sense to include either a sequence of actions P_Ω or a set of bindings B_Ω for the adversary. Instead, at any time, the adversary can receive any message or it can send any

[1] Principal and agent will be used synonymously throughout the paper

message it can generate from its set of known messages I_Ω. The global model is then simply the asynchronous composition of the models for each session, including the one corresponding to the adversary.

4 Actions

The actions allowed during the execution of a protocol include the two predefined actions **send** and **receive** as well as possibly some user defined actions. The model makes transitions between global states as a result of actions executed by the sessions. More formally, we define a transition relation $\rightarrow \subseteq \Sigma \times S \times A \times \mathcal{M} \times \Sigma$ where Σ is the set of global states, S again is the set of session IDs, A is the set of action names (which includes **send** and **receive**), and $\mathcal{M}$ is the set of all possible messages. We will use the notation $\sigma \xrightarrow{s \cdot a \cdot m} \sigma'$ in place of $(\sigma, s, a, m, \sigma') \in \rightarrow$ when it is more convenient. In the definitions below, we will denote the adversary's session as $\Omega = \langle N_\Omega, S_\Omega, \phi, I_\Omega, \emptyset \rangle$ and the sessions corresponding to the honest agents as $\Psi_i = \langle N_i, S_i, B_i, I_i, P_i \rangle$. We will use $\sigma = \langle \Omega, \Psi_1, \ldots, \Psi_n \rangle$ to denote the global state before the transition and $\sigma' = \langle \Omega', \Psi'_1, \ldots, \Psi'_n \rangle$ to denote the global state after the transition. In addition, we will use the notation $\hat{B}$ to denote the obvious extension of a set of bindings B from the domain of variables to the domain of message templates. In other words, $\hat{B}(m)$ is the result of substituting $B(v)$ for every occurrence of v in the message template m for all the variables v appearing in m.

- $\sigma \xrightarrow{s \cdot \mathbf{send} \cdot m} \sigma'$
  A session with ID s can send message m in global state σ and the new global state is σ' if and only if
  1. $I_{\Omega'} = I_\Omega \cup m$. (The adversary adds m to the set of messages it knows.)
  2. There is a session $\Psi_i = \langle N_i, s, B_i, I_i, \mathbf{send}(\textit{s-msg}).P'_i \rangle$ in σ such that in σ', $\Psi'_i = \langle N_i, s, B_i, I_i, P'_i \rangle$ and $m = \hat{B}_i(\textit{s-msg})$. (There is a session that is ready to send message m.)
  3. $\Psi_j = \Psi'_j$ for all $j \neq i$. (All other sessions remain unchanged.)
- $\sigma \xrightarrow{s \cdot \mathbf{receive} \cdot m} \sigma'$
  A session with ID s can receive message m in global state σ and the new global state is σ' if and only if
  1. $m \in \overline{I_\Omega}$. (The adversary can generate the message m.)
  2. There is a session $\Psi_i = \langle N_i, s, B_i, I_i, \mathbf{receive}(\textit{r-msg}).P'_i \rangle$ in σ such that in σ', $\Psi'_i = \langle N_i, s, B'_i, I'_i, P'_i \rangle$, $I'_i = I_i \cup m$, and B'_i is the smallest extension of B_i such that $\hat{B'}_i(\textit{r-msg}) = m$. (There is a session ready to receive a message of the form of m and its bindings are updated correctly in the next state.)
  3. $\Psi_j = \Psi'_j$ for all $j \neq i$. (All other sessions remain unchanged.)
- $\sigma \xrightarrow{s \cdot \mathbf{Act} \cdot m} \sigma'$
  A session with ID s can perform some user defined internal action $\mathbf{Act}$ with argument m in global state σ and the new global state is σ' if and only if

1. There is a session $\Psi_i = \langle N_i, s, B_i, I_i, \mathbf{Act}(msg).P'_i\rangle$ in σ such that in σ', $\Psi'_i = \langle N_i, s, B_i, I_i, P'_i\rangle$ and $m = \hat{B}_i(msg)$. (There is a session s that is ready to perform action $\mathbf{Act}$ with argument m.)
2. $\Psi_j = \Psi'_j$ for all $j \neq i$. (All other sessions remain unchanged).

Notice that internal actions are purely symbolic, i.e., there is no semantics associated with these actions.

Each possible execution of the model corresponds to a *trace*, a finite, alternating sequence of global states and actions $\pi = \sigma_0\alpha_1\sigma_1\alpha_2\cdots\alpha_n\sigma_n$ for some $n \in \mathbb{N}$, such that $\sigma_{i-1} \overset{\alpha_i}{\to} \sigma_i$ for $0 < i \leq n$ for the transition relation $\to$ just defined. Actually, technically speaking α_i belongs to the set $S \times A \times M$, but abusing the notation slightly we will refer to α_i as an action.

5 Logic

In order to specify the requirements or the desired properties of the protocol, we will use a first order logic where quantifiers range over the finite set of instances in a model. In addition, the logic will include the past-time modal operator so that we can talk about things that happened in the history of a particular protocol run or trace. The atomic propositions of the logic will allow us to refer to the bindings of variables in the model, to actions that occur during execution of the protocol, and to the knowledge of the different agents participating in the protocol. We will begin with the syntax of the logic, followed by the formal semantics.

5.1 Syntax

As stated above, we will use a first order logic where quantifiers range over the finite set of instances. The atomic propositions are used to characterize states, actions, and knowledge in the model. The arguments to the atomic propositions are terms expressing instances or messages. We begin by a formal description of terms.

– If S is a instance ID, then S is an instance term.
– If s is an instance variable, then s is an instance term.
– If M is a message, then M is a message term.
– If m is a message variable, then m is a message term.
– If s is an instance term, then $pr(s)$ represents the principal that is executing instance s.
– If s is an instance term and m is a message variable, then $s.m$ is a message term representing the binding of m in the instance s.
– If m_1 and m_2 are message terms, then $m_1 \cdot m_2$ is a message term.
– If m_1 and m_2 are message terms, then $\{m_1\}_{m_2}$ is a message term. Note that here we implicitly assume that m_2 is of atomic type key.

As in standard first order logic, atomic propositions are constructed from terms using relation symbols. The predefined relation symbols are "=" and "**Knows**". The user can also define other relation symbols which would correspond to user defined actions in the model. The syntax for atomic propositions is as follows: (All relation symbols are used in the infix notation.)

- If m_1 and m_2 are message terms, then $m_1 = m_2$ is an atomic proposition. Examples of this atomic proposition would be checking if a customer and merchant agree on the price of a purchase (C_0.*price* $= M_0$.*price*), or to check if a particular instance of A believes it's authenticating with B (A_0.*partner* = B).
- If s is an instance term and m is a message term, then s **Knows** m is an atomic proposition which intuitively means that instance s knows the message m. This proposition can be used to check if the adversary has compromised the session key (Ω **Knows** K)
- If s is an instance term, m is a message term, and **Act** is a user defined action, then s **Act** m is an atomic proposition which intuitively means that instance s performed action **Act** with message m as an argument. For example, this could be used to check if a customer C_0 has committed to a transaction with identifier *TID* (C_0**commit** *TID*).

Finally, *well-formed formulas* (or *wffs* for short) are built up from atomic propositions with the usual connectives from first-order and modal logic.

- if f is an atomic proposition, then f is a wff.
- if f is a wff, then $\neg f$ is a wff.
- if f_1 and f_2 are wffs, then $f_1 \wedge f_2$ is a wff.
- if f is a wff and s is an instance variable, then $\exists s.f$ is a wff.
- if f is a wff, then $\Diamond_P f$ is a wff.

The formula $\exists s.f$ has the intent that there exists some instance s_0 such that f is true when you substitute s_0 for s in f while $\Diamond_P f$ is supposed to mean that at some point in the past, f was true. We also use the following common shorthands:

- $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$
- $f_1 \leftrightarrow f_2 \equiv f_1 \rightarrow f_2 \wedge f_2 \rightarrow f_1$
- $\forall s.f \equiv \neg\exists s.\neg f$ (For all instances, s_0, f is true when you substitute s_0 for s.)
- $\Box_P f \equiv \neg\Diamond_P \neg f$ (At all points in the past, f was true.)

The formula $\forall s.f$ is supposed to mean that for any instance s_0, f is true when you substitute s_0 for s in f while $\Box_P f$ is supposed to mean that at all points in the past, f was true.

5.2 Semantics

Next we provide semantics to the logic just presented. These semantics will be given in terms of the formal model presented in Section 3. Again, we begin with the terms of the logic.

- An instance ID S refers to the instance with that ID.
- An instance variable s ranges over all the instances corresponding to the honest agents in the model.
- An atomic message M is an atomic message in the model.

- A message variable v varies over messages in the model and can be defined as a binding variable in a particular principal.
- The function pr maps an instance ID to a principal name. If s is an instance ID, then $pr(s)$ is the principal executing the instance with ID s.
- We use "." as a scoping operator. If s is an instance term and v is a message variable, then $s.v$ refers to the variable v bound in the instance s. The interpretation $\sigma(s.v)$ of $s.v$ in a particular state σ is $B_s(v)$, the value bound to the variable v in instance s in state σ.
- Message terms can be concatenated using "$\cdot$" just as messages are concatenated.
- Similarly a message term m_1 can be encrypted with another message term m_2 just as messages are encrypted in the model.

The wffs of the logic will be interpreted over the traces of a particular model. Recall that a trace consists of a finite, alternating sequence of states and actions $\pi = \sigma_0 \alpha_1 \sigma_1 \ldots \sigma_n$. Length of a trace π is denoted by *length*(π). We give the semantics of wffs in our model via a recursive definition of the satisfaction relation $\models$. We will write $\langle \pi, i \rangle \models f$ to mean that the i-th state in π satisfies the formula f. We begin with atomic propositions.

- $\langle \pi, i \rangle \models m_1 = m_2$ iff $\sigma_i(m_1) = \sigma_i(m_2)$. Thus the formula $m_1 = m_2$ is true in a state if the interpretations of m_1 and m_2 are equal. In other words, two message terms are equal in a state if after applying the appropriate substitutions to the variables appearing in the message terms, the resulting messages are equal.
- The formula $\langle \pi, i \rangle \models s \ \mathbf{Knows}\ m$ iff $\sigma_i(m) \in \overline{I_j}$ for some instance Ψ_j in σ_i such that $S_j = s$ (the instance ID of Ψ_j is s). In other words, the formula $s\ \mathbf{Knows}\ m$ is true in a state if the instance with ID s can derive message m from its known set of messages in that state. $\Omega\ \mathbf{Knows}\ m$ is true if the adversary Ω knows message m (recall that Ω denotes the adversary).
- $\langle \pi, i \rangle \models s\ \mathbf{Act}\ m$ for some user defined action $\mathbf{Act}$ iff $\alpha_i = s \cdot \mathbf{Act} \cdot m$. In other words, the formula $(s\ \mathbf{Act}\ m)$ is true in a state if the transition taken to enter the current state was one in which instance s took action $\mathbf{Act}$ with argument m.

The extension of the satisfaction relation to the logical connectives is the same as for standard first order logic. We use the notation $f[s_0 \backslash s]$ to denote the result of substituting every free occurrence of the instance variable s in f with the instance ID s_0.

- $\langle \pi, i \rangle \models \neg f$ iff $\langle \pi, i \rangle \not\models f$.
- $\langle \pi, i \rangle \models f_1 \wedge f_2$ iff $\langle \pi, i \rangle \models f_1$ and $\langle \pi, i \rangle \models f_2$
- $\langle \pi, i \rangle \models \exists s.f$ iff there exists a honest instance s_0 in the model such that $\langle \pi, i \rangle \models f[s_0 \backslash s]$.
- $\langle \pi, i \rangle \models \Diamond_P f$ iff there exists a $0 \leq j \leq i$ such that $\langle \pi, j \rangle \models f$ In other words, the formula $\Diamond_P f$ is true in a state of a trace π if the formula f is true in any state of the trace up to and including the current state.

A formula f is said to be true in a trace π (denoted as $\pi \models f$) iff f is true in *every state* of the trace π.

5.3 Specification examples

For the sake of concreteness, we now include examples of some of the properties we have checked using BRUTUS and how they are specified in our logic. For the sake of clarity, we break the specification into two parts. The first part (referred to as ϕ_H) expresses properties about honest agents. The second part (referred to as ϕ_Ω) pertains to the adversary. Hence, the entire specification ϕ is simply $\phi_H \wedge \phi_\Omega$.

Payment Authorization. For the secure payment 1KP protocol [BGH+95], we wish to show that whenever the customer's account is debited, the customer must have authorized that debit. For this we simply choose ϕ_H to be

$$\forall A_0 \,.\, (pr(A_0) = A) \wedge (A_0 \textbf{ debit } (A_0.\text{CC} \cdot A_0.\text{price})) \rightarrow$$

$$\exists C_0.(pr(C_0) = C) \wedge (A_0.\text{CC} = C_0.\text{CC}) \wedge \Diamond_P(C_0 \textbf{ auth } A_0.\text{price})$$

This formula states that for all sessions A_0, if A_0 is a session being executed by the authority A, and A_0 debits the credit card account A_0.CC by A_0.price, then there exists a session C_0 being executed by the customer C with that same credit card number that authorized a debit of that amount. Since in this case we do not refer to the adversary, we let $\phi_\Omega = \textbf{true}$.

Privacy. The 1KP protocol should not reveal information about the transaction. In other words, only the appropriate principals should know the order information. For this we choose ϕ_H to be

$$\forall S_0 \,.\forall C_0 \,.\, (pr(C_0) = C) \wedge \ (S_0 \textbf{ Knows } C_0.\textit{DESC}) \rightarrow$$

$$(pr(S_0) = C \vee pr(S_0) = M)$$

This formula states that for all sessions S_0, if S_0 knows the customer's description (C_0.*DESC*) of the transaction, then S_0 is a session being executed by either the customer or the merchant. We will also need to make sure that the adversary does not know the information, so we choose ϕ_Ω to be

$$\forall C_0 \,.\, (pr(C_0) = C) \rightarrow \neg(\Omega \textbf{ Knows } C_0.\textit{DESC})$$

Non-repudiation. We may want to check that a principal cannot deny knowledge of a particular value (a key or nonce). For instance, in the Needham-Schroeder authentication protocol [NS78], we may want to make sure that whenever A ends a session with B, B must know the nonce created by A. Note, that this is a somewhat weak notion of non-repudiation. A may not be able to prove B's knowledge of A's nonce. Indeed, A may not even be able to convince itself that B knows the nonce. We are simply checking that there is no trace in which B does not know the nonce. For this specification we choose ϕ_H to be

$$\forall S_0 \,.\forall T_0.\ S_0 \textbf{ end } \ pr(T_0) \rightarrow (T_0 \textbf{ Knows } S_0.\textit{Nonce})$$

This formula states that for all pairs of sessions S_0 and T_0, if S_0 ends a session with T_0, then T_0 knows the nonce generated by S_0. We choose ϕ_Ω = **true**.

5.4 An ordering on traces

We introduce an ordering on traces that will aid us in proving the correctness of partial order reductions. Throughout this sub-section assume that we are given a specification ϕ. Since the number of sessions is finite, we can assume that the specification is quantifier free (see the equations given below).

$$\begin{array}{rcl} \exists s.f & = & \vee_{i=1}^{n} f[s_i \backslash s] \\ \forall s.f & = & \wedge_{i=1}^{n} f[s_i \backslash s] \end{array}$$

In the equations given above we have assumed that there are n honest sessions with session IDs $s_1, \cdots, s_n$. We also assume that the negations are pushed down to the innermost level. A quantifier free formula where the negation has been pushed to the innermost level is said to be in *negation normal form*. Let AP_H be the set of atomic formulas corresponding to honest sessions that appear in the specification (see subsection 5.1). Similarly, let AP_Ω be the set of atomic formulas pertaining to the adversary that appear in the specification. A specification is called *admissible* if it is in negation normal form and the atomic formulas in AP_Ω appear negated. From here on, assume that formulas are constructed using the set of atomic formula $AP_H \cup AP_\Omega$, and are admissible. We let $\mathcal{CF}$ be the class of normal and admissible formula built using the atomic formula in the set $AP_H \cup AP_\Omega$.

Notice that the truth of a specification ϕ on a trace π is completely determined by the values of the atomic propositions in the set AP_H and the adversary's knowledge (the set of messages known to the adversary) at each state of the trace π. Adversary's knowledge determines the truth of the atomic formula in the set AP_Ω. Assume that for each state σ we are given the set of atomic propositions true in that state and the knowledge of the adversary. $L(\sigma) \subseteq 2^{AP_H}$ is the labelling function which indicates whether an atomic proposition in AP_H is true in the state σ or not ($p \in L(\sigma)$ means that the atomic proposition p is true in the state σ). Knowledge of the adversary in state σ is denoted by $I_\Omega(\sigma)$, or equivalently the set of messages known to the adversary in the state σ is $I_\Omega(\sigma)$. We introduce a partial order between traces, which will help us to prove the correctness of our reduction techniques.

Definition 1. A trace π_1 is *greater than* a trace π_2 (denoted by $\pi_1 \succeq \pi_2$) iff there exists partitions $\{A_1, \cdots, A_m\}$ and $\{B_1, \cdots, B_m\}$ of the two traces π_1 and π_2 such that the following conditions hold:

– There exists $0 = a_0 < a_1 < a_2 < \cdots < a_m = \mathit{length}(\pi_1)$ such that $A_k = \{\langle \pi_1, a_{k-1}+1 \rangle, \cdots, \langle \pi_1, a_k \rangle\}$, or in other words A_k represents the sub-trace starting at index $a_{k-1}+1$ and ending at a_k.

– A symmetric condition holds for the partition $\{B_1, \cdots, B_m\}$ of the trace π_2 with indices $0 = b_0 < b_1 < b_2 < \cdots < b_m = \mathit{length}(\pi_2)$.

– For two states in the corresponding partitions A_k and B_k ($1 \leq k \leq m$) the labelling of atomic propositions in AP_H is identical and adversary's knowledge in *every* state of A_k is greater than in the last state of B_k. Since knowledge of the adversary is monotonic along a trace (adversary never forgets anything), this also implies that the adversary's knowledge in an arbitrary state of A_k is more than the knowledge in all states of B_k. More precisely, the following conditions hold:

$$\forall s \in A_k \forall s' \in B_k (L(s) = L(s'))$$
$$\forall s \in A_k (I_\Omega(s) \supseteq I_\Omega(\langle \pi_2, b_k \rangle))$$

Informally, the lemma given below states that if $\pi_1 \succeq \pi_2$ the correctness of an admissible specification in π_1 implies its correctness in π_2.

Lemma 1. Given two traces π_1, π_2 such that $\pi_1 \succeq \pi_2$ and an admissible specification ϕ, $\pi_1 \models \phi$ implies that $\pi_2 \models \phi$. In other words, the partial order $\succeq$ is monotonic with respect to the satisfaction relation $\models$.

6 Partial Order Reduction

Partial order reductions reduce the search space by ignoring redundant interleavings. The theory of partial order reductions is well developed in the context of verification of reactive systems [GPS96,Pel96,Val91]. Reductions presented in this section are very heavily influenced by traditional partial order reduction techniques. However, since we are working with a very specific model and logic, the theory is simplified and different. We present the theory as it applies to our setting.
Throughout this section assume that we are given a specification ϕ that is admissible. Recall that AP_H is the set of atomic propositions pertaining to the honest agents and AP_Ω is the set of atomic propositions referring to the adversary. An internal action $\mathbf{Act}$ is called *invisible* if and only if it does not appear in the specification ϕ, or in other words $\mathbf{Act}$ is not referred to by the atomic formulas in the set AP_H. Next we describe transformations on traces.

Permuting invisible internal actions (Rule 1)
Consider a trace $\pi = \sigma_0 \alpha_1 \sigma_1 \ldots \sigma_n$. If there exists a sequence of transitions $\sigma_i \alpha_{i+1} \sigma_{i+1} \alpha_{i+2}$, such that α_{i+2} is an invisible internal action, and actions α_{i+1} and α_{i+2} do not belong to the same session, then we can permute the actions to get a new trace given below:

$$\sigma_0 \alpha_1 \sigma_1 \ldots \sigma_i \alpha_{i+2} {\sigma'}_{i+1} \alpha_{i+1} \ldots \sigma_n$$

Permuting sends (Rule 2)
This operation allows one to permute two consecutive $\mathbf{send}$ actions if they belong to different sessions.

Moving send before receives (Rule 3)
If a $\mathbf{receive}$ or an internal action $\mathbf{Act}$ appears before a $\mathbf{send}$ in a trace and these actions belong to different sessions, then this operation allows us to move the $\mathbf{send}$ action before the $\mathbf{receive}$ or the internal action $\mathbf{Act}$.

We call the set of transformations just described *allowable operations* on a trace. Suppose we obtain a trace π' by applying one of the *allowable operations* to the trace π, then we say that $\pi \Rightarrow \pi'$. The reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^\star$. The following lemma is crucial in proving correctness of the partial order reduction.

Lemma 2. *Consider two traces π and π' such that $\pi \Rightarrow \pi'$. In this case $\pi \preceq \pi'$.*

Using Lemmas 2 and 1, the proof of the following lemma is transparent (note that $\pi \preceq \pi'$).

Lemma 3. *Assume that we are given a specification ϕ. If there are two traces π and π' such that $\pi \Rightarrow^\star \pi'$, then $\pi' \models \phi$ implies that $\pi \models \phi$, or equivalently $\pi \not\models \phi$ implies that $\pi' \not\models \phi$.*

The basic algorithm for verifying whether a protocol satisfies a specification works by exploring the state space starting from the initial state using depth-first search. As soon as we reach a state where the specification is false, we report an error. If the depth-first search procedure terminates without reporting an error, the protocol is correct. In the ensuing discussion we will focus on the depth-first search algorithm. In the description of the algorithms we do not show book-keeping details such as reporting an error or checking whether a state has been visited or not. Algorithm $\mathcal{A}$ given in Figure 1 performs the depth-first search starting from state s. The predicate $en(s, \alpha)$ is true if action α is enabled in the state s, i.e., action α can be executed from the state s. The set of enabled actions $EN(s)$ in a state s is $\{ \alpha \mid en(s, \alpha) \}$. Algorithm $\mathcal{A}_{\mathcal{PO}}$ (shown in Figure 2) is the modified depth-first search procedure with partial-order reductions. The set of actions $ample(EN(s))$ is defined as follows:

- If $EN(s)$ contains an invisible internal action , then $ample(EN(s))$ is an arbitrary invisible action $\{\textbf{Act}\}$ picked from $EN(s)$.
- Suppose $EN(s)$ does not contain an invisible internal action, but does contain a $\mathbf{send}$ action. In this case $ample(EN(s))$ is an arbitrary $\mathbf{send}$ action picked from the set $EN(s)$.
- If $EN(s)$ does not contain an invisible internal action or a $\mathbf{send}$ action, $ample(EN(s))$ is equal to $EN(s)$

Theorem 1 proves the correctness of the partial order reduction. Notice that the reduced algorithm $\mathcal{A}_{\mathcal{PO}}$ explores fewer traces than the algorithm $\mathcal{A}$. Theorem 1 basically states that every trace considered by the exhaustive algorithm $\mathcal{A}$ can be transformed into a trace considered by the reduced algorithm $\mathcal{A}_{\mathcal{PO}}$ using allowable operations described earlier.

```
funct dfs($s$)
  EN (s) = $\{ \alpha \mid en(s, \alpha) \}$
  foreach $\alpha \in$ EN (s)
    do dfs($\alpha(s)$)
```

Fig. 1. Depth first search algorithm $\mathcal{A}$

Theorem 1. *For every trace π considered by the algorithm $\mathcal{A}$, algorithm $\mathcal{A}_{\mathcal{PO}}$ considers a trace π' such that $\pi \Rightarrow^\star \pi'$.*

```
funct $dfs(s)$
  $EN(s) = \{ \alpha \mid en(s, \alpha) \}$
  foreach $\alpha \in ample(EN(s))$
    do $dfs(\alpha(s))$
```

Fig. 2. Modified depth first search algorithm $\mathcal{A}_{\mathcal{PO}}$

Using this theorem along with other results proved earlier, subsequent discussion shows that the algorithm with partial order reduction will discover an incorrect trace if and only if the full algorithm $\mathcal{A}$ will discover an incorrect trace. Suppose the protocol we are verifying is incorrect. In this case algorithm $\mathcal{A}$, being exhaustive in nature, will consider a trace π such that $\pi \not\models \phi$. Using Theorem 1 we can deduce that the reduced algorithm $\mathcal{A}_{\mathcal{PO}}$ considers a trace π' such that $\pi \Rightarrow^{\star} \pi'$. Using Lemma 3 we obtain that $\pi' \not\models \phi$. Therefore, if the protocol is incorrect, the reduced algorithm will detect it. Since the reduced algorithm only executes a subset of actions enabled from a state, it only considers a subset of the entire set of traces. This means that if the reduced algorithm finds an incorrect trace, the protocol is incorrect. Hence the protocol is *correct if and only the reduced algorithm $\mathcal{A}_{\mathcal{PO}}$ does not find an incorrect trace*. Therefore, the reduced algorithm $\mathcal{A}_{\mathcal{PO}}$ can be safely used.

7 Experimental Results

The table shown in Figure 7 summarizes the results of applying partial reductions to a few protocols. We examined the 1KP secure payment protocol [BGH$^+$95], the Needham-Schroeder public key protocol [NS78], and the Wide-Mouthed Frog protocol [BAN90,Sch96]. Columns 2 and 3 give the number of initiator and responder sessions used in the model. The other columns give the number of states encountered during state space traversal using exhaustive search and search with partial order reductions. The entries with an "X" represents computations that were aborted after a day of computation (over 700,000,000 states).

protocol	init	resp	none	partial order
1KP	1	1	17,905,267	906,307
N-S	1	1	1,208	146
N-S	1	2	1,227,415	6,503
WMF	3	3	X	1,286,074

Fig. 3. Table of results

8 Related Work

As mentioned in the introduction, there are several research efforts that have applied existing model checkers to the verification of security protocols. Our model checker is especially built to check properties of cryptographic and electronic commerce protocols. For example, we explicitly keep track of the knowledge for each agent and our logic can refer to the knowledge of various agents. However, because we extend the system state to keep track of knowledge, the correctness of various reduction techniques in the domain of traditional model checking cannot be directly applied. Here we developed the theory of partial order reductions for the verification of cryptographic and electronic commerce protocols. A reduction similar to partial order reduction appears in [SS98]. In [SS98] authors use Murϕ to verify cryptographic protocols. The connection to partial order reductions was not made in [SS98] and the set of reductions considered in [SS98] are more restrictive than the ones considered here. Moreover, the arguments presented in [SS98] only apply to a restrictive logic. Arguments presented in this paper are much more precise and apply to a much richer logic.

9 Conclusion

In this paper we presented a logic for specifying properties of security protocols. In this context, we also presented partial order reduction techniques. Experimental results clearly indicate that this reduction technique significantly reduces the size of the state space. In the future, we want to test our ideas on larger protocols. Currently, internal actions do not have any semantics associated with them. In the future we also want to add semantics to internal actions, e.g., the **debit** action will actually debit the customer's account.

Acknowledgements

Darrell Kindred, Juergen Dingel, and Helmut Veith provided very helpful comments that enhanced the overall quality of the paper.

References

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[BGH$^+$95] M. Bellare, J. Garay, R. Hauser, A. Herberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP - a family of secure electronic payment protocols. In *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, July 1995.

[BP97] G. Bella and L. C. Paulson. Using isabelle to prove properties of the kerberos authentication system. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[CJM98] E. M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[GPS96] Patrice Godefroid, Doron Peled, and Mark Staskauskas. Using partial order methods in the formal validation of industrial concurrent programs. In *ISSTA'96, International Symposium on Software Testing and Analysis*, pages 261–269, San Diego, California, USA, 1996. ACM Press.

[KW97] D. Kindred and J. M. Wing. Closing the idealization gap with theory generation. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[Low97] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 18–30, 1997.

[Mea96] C. Meadows. Language generation and verification in the NRL protocol analyzer. In *Proceedings of the 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pel96] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Journal of Formal Methods in Systems Design*, 8 (1):39–64, 1996. also appeared in 6th International Conference on Computer Aided Verification 1994, Stanford CA, USA, LNCS 818, Springer-Verlag, 377-390.

[Ros96] A. W. Roscoe. Intensional specifications of security protocols. In *9th Computer Security Foundations Workshop*, 1996.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.

[SS98] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of the 1998 Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1998.

[Val91] A. Valmari. Stubborn sets of colored petri nets. In *Proceedings of the 12th International Conference on Application and Theory of Petri Nets*, pages 102–121, Gjern, Denmark, 1991.

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.