Higher-order quantification and proof search

Extended abstract

Dale Miller, Penn State University
Sept 2002: INRIA and École Polytechnique

Outline

1. Security protocols, multisets rewriting, linear logic.
2. Higher-order quantifier: encrypted data as an abstract datatype.
3. Higher-order quantifier: hiding names for continuations
4. Asynchronous, synchronous, and bipolar formulas

Full version of paper: AMAST 2002, 9-13 September 2002, Reunion Island.

A Typical Protocol Specification

The following is a presentation of the Needham-Schroeder Shared Key Protocol. Alice and Bob make use of a trusted server to help them establish their own private channel for communications.

Message 1	$A \longrightarrow S\colon\ A, B, n_A$
Message 2	$S \longrightarrow A\colon\ \{n_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$
Message 3	$A \longrightarrow B\colon\ \{k_{AB}, A\}_{k_{BS}}$
Message 4	$B \longrightarrow A\colon\ \{n_B\}_{k_{AB}}$
Message 5	$A \longrightarrow B\colon\ \{n_B - 1\}_{k_{AB}}$

Here, A, B, and S are agents (Alice, Bob, server), and the k's are encryption keys, and the n's are nonces.

One of our goals is to replace this specific syntax with one that is based on a direct use of logic. We will then investigate if logic's meta-theory can help in reasoning about security.

Motivating a more declarative specification

The notation $A \longrightarrow B\colon M$ seems to indicate a “three-way synchronization”, but communication here is asynchronous: Alice put a message on in a network and Bob picks it up from the network. An intruder might get the message and read and/or delete it.

Better seems to be a syntax like:

$$\begin{aligned} A \quad &\longrightarrow A' \mid M \\ B \mid M &\longrightarrow \quad B' \\ &\vdots \\ E \mid M &\longrightarrow E' \mid M \end{aligned}$$

More generally,

$$A \mid M_1 \mid \cdots \mid M_p \longrightarrow A' \mid N_1 \mid \cdots \mid N_q$$

where $p, q \geq 0$. One occurrence of the agent could be missing from the left (i.e., agent creation) or one can be missing from the right (i.e., agent deletion).

Execution of such specifications can be considered as **multiset rewriting**.

Data, network messages, agent memory

All communicated items are given the type *data*.

$\langle\rangle$	unit, empty pair, nil
$\langle x, y\rangle$	pair, cons
$\langle x_1, \ldots, x_n\rangle$	pairing associated to the right

Integers, strings, nonces, etc, will be considered part of this one type. This may not be realistic: richer typing can be accommodated if necessary. Doing so, however, is not central to this talk.

A **network message** is an atomic formula $[[M]]$ where $[[\cdot]]$ is a predicate with an argument of type *data*.

The **state** of an agent is encoded as an atomic formulas, using a predicate of one argument: the argument encodes its **memory**.

$$(\mathit{Agent}_i\ \mathit{Memory}) \mid M_1 \mid \cdots \mid M_p \longrightarrow (\mathit{Agent}_{i+1}\ \mathit{Memory'}) \mid N_1 \mid \cdots \mid N_q$$

Creation of new symbols

New symbols representing nonces (used to help guarantee “freshness”) and new keys for encryption and session management are needed also in protocols. We could introduced syntax such as:

$$a_1 \ S \longrightarrow \mathit{new}\ k[\ a_2\ \langle k, S\rangle \mid [[\{M\}_k]]\ ]$$

This *new* operator looks a bit like a quantifier: it should support α-conversion and seems to be a bit like reasoning generically. The scope of *new* is over the body of this rule.

Static distribution of keys

Consider a protocol containing the following messages.

$$\vdots$$

Message i	$A \longrightarrow S$: $\{M\}_k$
Message j	$S \longrightarrow A$: $\{P\}_k$

$$\vdots$$

How can we declare that a key, such as k, is only built into two specific agents. This static declaration is critical for modularity and for establishing correctness later. A **local** declaration can be used.

$$\vdots$$

$$\mathit{local}\ k.\ \left\{\begin{array}{r} A \longrightarrow A' \mid\ [[\{M\}_k]] \\ S \mid\ [[\{P\}_k]] \longrightarrow S' \end{array}\right\}$$

$$\vdots$$

This declarations also appears to be similar to a quantifier.

Can we see our specifications as being logic?

Can we view the symbols we have introduced as logical connectives? In general, we would not expect this, but if it is possible, there might be significant benefits from this change of perspective.

syntax	$\mid$	$\longrightarrow$	*new*	*local*	*empty*
disjunctive	$\parr$	$\multimapinv$	$\forall$	$\exists$	$\perp$
conjunctive	$\otimes$	$\multimap$	$\exists$	$\forall$	$\mathbf{1}$

The disjunctive approach allows this to fit into the "logic programming as goal-directed search" paradigm and as a subset of the Forum presentation of linear logic.

Encrypted data as an abstract data type

The standard logic programming approach to abstract data types can be used to capture encrypted data.

Encryption keys are coded as symbolic functions on data of type $data \rightarrow data$ and they will be provided scope via the use of the local and new declarations. We replace $\{M\}_k$ with just $(k\ M)$.

To insert an encryption key into data, we will use the postfix coercion constructor $(\cdot)^\circ$ of type $(data \rightarrow data) \rightarrow data$.

The use of higher-order types means that we will also use the equations of $\alpha\beta\eta$-conversion (a well studied extension to logic programming with robust implementations).

$$\exists\ k.\ \left[\begin{array}{rl} a_1\ S \multimap & \forall n.\ a_2\ \langle k^\circ, S\rangle \parr [[k\ n]] \\ a_2\ \langle k^\circ, S\rangle \parr [[(k\ M)]] \multimap & \ldots \end{array}\right]$$

A Linear Logic Specification of Needham-Schroeder

$$
\begin{array}{lll}
\multicolumn{3}{l}{\exists k_{as} \exists k_{bs} \{} \\
\quad a\ S & \multimapinv \forall na. & a_1\ \langle na, S\rangle \parr [[\langle a, b, na\rangle]]. \\
\quad a_1\ \langle N, S\rangle \parr [[(k_{as}\langle N, b, K, En\rangle]] & \multimapinv & a_2\ \langle N, K, S\rangle \parr [[En]]. \\
\quad a_2\ \langle Na, Key^\circ, S\rangle \parr [[(Key\ Nb]]) & \multimapinv & a_3\ \langle\rangle \parr [[(Key\ \langle Nb, S\rangle)]]. \\
\quad b\ \langle\rangle \parr [[(k_{bs}\ \langle Key^\circ, a\rangle]] & \multimapinv \forall nb. & b_1\ \langle nb, Key^\circ\rangle \parr [[(Key\ nb)]]. \\
\quad b_1\ \langle Nb, Key\rangle \parr [[(Key\langle Nb, S\rangle)]] & \multimapinv & b_2\ S. \\
\quad s\ \langle\rangle \parr [[\langle a, b, N\rangle]] & \multimapinv \forall k. & s\ \langle\rangle \parr [[(k_{as}\langle N, b, k^\circ, k_{bs}\langle k^\circ, a\rangle\rangle)]]. \\
\}
\end{array}
$$

When “executed via proof search,” all higher-order quantification is essentially trivial: either generates an eigenvariable or is instantiated with an eigenvariable.

Outermost universal quantifiers around individual clauses have not been written but are assumed for variables (tokens starting with a capital letter).

Relating implementation and specification

A property of NS should be that Alice can communicate to Bob a secret with the help of a server. That is, the clause

$$\forall x\ [a\ \langle x\rangle \parr b\ \langle\rangle \parr s\ \langle\rangle \multimapinv a_3\ \langle\rangle \parr b_2\ \langle x\rangle \parr s\ \langle\rangle]$$

can be seen as part of the specification of this protocol.

If we call the above clause *SPEC* and the formula for Needham-Schroeder *NS*, then it is a simple calculation to prove that $NS \vdash SPEC$ in linear logic.

Of course, a kind of converse is more interesting and harder. At least a trivial thing is proved trivially.

Should not logical entailment be a center piece of logical specifications?

A simple logical equivalence

Consider the following two clauses:

$$a \multimap \forall k.[[(k\ m)]] \quad \text{and} \quad a \multimap \forall k.[[(k\ m')]].$$

These two clause show that Alice can take a step that generates a new encryption key and then outputs either the message m or m' in encrypted form. These two clauses seem “observationally similar”.

More surprisingly

$$a \multimap \forall k.[[(k\ m)]] \dashv\vdash a \multimap \forall k.[[(k\ m')]].$$

That is, they are logically equivalent! In particular, the sequent

$$\forall k.[[(k\ m)]] \longrightarrow \forall k.[[(k\ m')]]$$

is proved by using the eigenvariable c on the right and the term $\lambda w.(c\ m')$ on the left.

More logical equivalences

If we allow local ($\exists$) abstractions of predicates, then other more interesting logical equivalences are possible.

For example, 3-way synchronization can be implemented using 2-way synchronization with a hidden intermediary.

$$\exists\, x. \left\{ \begin{array}{l} a \parr b \multimapinv x \\ x \parr c \multimapinv d \parr e \end{array} \right\} \quad \dashv\vdash \quad a \parr b \parr c \multimapinv d \parr e$$

Intermediate states of an agent can be taken out entirely.

$$\exists\, a_2, a_3. \left\{ \begin{array}{l} a_1 \parr [[m_0]] \multimapinv a_2 \parr [[m_1]] \\ a_2 \parr [[m_2]] \multimapinv a_3 \parr [[m_3]] \\ a_3 \parr [[m_4]] \multimapinv a_4 \parr [[m_5]] \end{array} \right\} \quad \dashv\vdash$$

$$a_1 \parr [[m_0]] \multimapinv ([[m_1]] \multimapinv ([[m_2]] \multimapinv ([[m_3]] \multimapinv ([[m_4]] \multimapinv ([[m_5]] \parr a_4)))))$$

This suggests an alternative syntax for agents.

Needham-Schroeder revisited

$$\begin{array}{ll}
\text{(Out)} & \forall na.[[\langle alice,\ bob,\ na\rangle]]\multimapinv \\
\text{(In)} & \quad(\forall Kab\forall En.[[kas\langle na,\ bob,\ Kab^\circ,\ En\rangle]]\multimapinv \\
\text{(Out)} & \qquad([[En]]\multimapinv \\
\text{(In)} & \qquad\quad(\forall Nb.[[(Kab\ Nb)]]\multimapinv \\
\text{(Out)} & \qquad\qquad[[(Kab(Nb,\ secret))]]))).
\end{array}$$

$$\begin{array}{ll}
\text{(Out)} & \bot\multimapinv \\
\text{(In)} & \quad(\forall Kab.[[(kbs(Kab^\circ,\ alice))]]\multimapinv \\
\text{(Out)} & \qquad(\forall nb.[[(Kab\ nb)]]\multimapinv \\
\text{(In)} & \qquad\quad([[(Kab(nb,\ secret))]]\multimapinv \\
\text{(Cont)} & \qquad\qquad b\ secret))).
\end{array}$$

$$\begin{array}{ll}
\text{(Out)} & \bot\Leftarrow \\
\text{(In)} & \quad(\forall N.[[\langle alice,\ bob,\ N\rangle]]\multimapinv \\
\text{(Out)} & \qquad(\forall key.[[kas\langle N,\ bob,\ key^\circ,\ kbs(key^\circ,\ alice)\rangle]])).
\end{array}$$

Two classes of connectives

The logical connectives of linear logic can be classified as

asynchronous $\bot$, $⅋$, $\forall$, …. The right introduction rules for these are invertible. These rules yield structural equivalences.

synchronous $\mathbf{1}$, $\otimes$, $\exists$, …. The right introduction rules for these are not invertible. These rules yield interaction with the environment.

These connectives are de Morgan duals of each other. For example, if an asynchronous connectives appears on the left of the sequent arrow, it acts synchronously.

We shall only write asynchronous connectives but write them on both sides of the sequent arrow (yielding both behaviors). We also use implications:

$$B \multimap C \equiv B^{\bot} ⅋ C \qquad \text{and} \qquad B \Rightarrow C \equiv\, !B \multimap C$$

Alternation of synchronous and asynchronous connectives

A *bipolar* formula is a formula in which no asynchronous connectives is in the scope of a synchronous connective. That is, there is an outer layer of asynchronous connectives followed by an inner layer of synchronous connectives.

The multiset rewriting clauses are bipolars, for example,

$$a \parr b \multimapinv c \parr d \equiv a \parr b \parr (c^\perp \otimes d^\perp).$$

Andreoli showed how to compile arbitrary alternation of syn/asyn connectives into bipolars by introducing new predicate symbols. He also argued for only using bipolars for proof search.

Avoiding bipolars has some advantages

Only one predicate is need, namely, $[\![\cdot]\!]$. The other predicates (used as "line numbers" in a protocol) are not needed.

The scope of variables within a formula encodes an agent's memory.

Agents now look much more like process calculus expressions with input and output prefixes. The formula $a \circ\!- (b \circ\!- (c \circ\!- (d \circ\!- k)))$ can denote either

$$\bar{a} \,||\, (b.\ (\bar{c} \,||\, (d.\ \ldots))) \quad \text{or} \quad a.\ (\bar{b} \,||\, (c.\ (\bar{d} \,||\, \ldots)))$$

depending on if it appears on the right or the left of the sequent arrow. Writing it and its negation without linear implications:

$$a \parr (b^{\perp} \otimes (c \parr (d^{\perp} \otimes \ldots))) \quad \text{resp,} \quad a^{\perp} \otimes (b \parr (c^{\perp} \otimes (d \parr \ldots)))$$

Value passing, name generation, and scope extrusion (ie, dynamic distribution of nonces and keys) are available.

There is a strict alternation of input and output phases. If an agent skips a phase, the adjacent phases can be merged:

$$a \circ\!- (\perp \circ\!- (b \circ\!- k)) \equiv (a \parr b) \circ\!- k.$$

The general setting for specifying agents

$$A = \text{ atomic formulas}$$

$$H = A \mid \bot \mid H \parr H \mid \forall x.\ H$$

$$K = H \mid H \circ\!\!- K \mid \forall x.\ K$$

Let $\mathcal{A}$ denote a multiset of atoms (ie, network messages). Let Γ and Δ be a multiset of "agents" (K-formulas). Since Γ will appear on the right, it contains outputting agents and since Δ will appear on the left, it contains inputting agents.

The two rules involving proof search with agents are then given as follows:

$$\frac{\Delta, K \longrightarrow \Gamma, H, \mathcal{A}}{\Delta \longrightarrow H \circ\!\!- K, \Gamma, \mathcal{A}} \qquad \frac{H \longrightarrow \mathcal{A}_1 \qquad \Delta \longrightarrow \Gamma, K, \mathcal{A}_2}{\Delta, H \circ\!\!- K \longrightarrow \Gamma, \mathcal{A}_1, \mathcal{A}_2}$$

If in the definition of K-formulas above we write $H \circ\!\!- H$ instead of $H \circ\!\!- K$, we are restricting our selves to bipolars again.

Conclusions

1. Linear logic can be used to specify the *execution* of this level of abstraction for security protocols. See: Cervesato, Durgin, Lincoln, Mitchell, and Scedrov in “A meta-notation for protocol analysis” [CSFW, June 1999].
2. Encryption can be seen as an abstract datatype.
3. Multiset rewriting and an asynchronous processes calculi are essentially equivalent modulo the treatment of agent continuation.
4. To what extent can common proof theoretical techniques be used to reason about protocol correctness issues?
   (a) Cut and cut-elimination are basic tools.
   (b) Higher-type quantification make protocols more declarative and offer new avenues for reasoning about protocols; they may not make proof search more complicated.
   (c) Induction and fixed points (definitions) will certainly be needed to strength reasoning further.