Verification of NSLPK

Lecture Note 09
Formal Methods (i613-0912)

Topics

- Brushup of the previous lecture
- Verification that (an abstract model of) NSLPK enjoys Agreement property
- proof score templates
- Case analysis & lemma conjecture
- Simultaneous induction

Brushup (1)

- NSLPK:

Init:	$p \rightarrow q$	$\{ n_p, p \}_{\mathrm{k}(q)}$
Resp:	$q \rightarrow p$	$\{ n_p, n_q, q \}_{\mathrm{k}(p)}$
Ack:	$p \rightarrow q$	$\{ n_q \}_{\mathrm{k}(q)}$

- Agreement Property: Whenever a protocol run is successfully completed by p and q,
  - the principal with which p is communicating is really q, and
  - the principal with which q is communicating is really p.

Brushup (2)

- Nonces: $\mathtt{n}(p, q, r)$ denotes a nonce made by p for q, where r makes it unique and unguessable.
- Messages: $\mathtt{m}i(p?, p, q, e_i)$ ($i = 1,2,3$) denotes a message (an Init, Resp, or Ack message) that seems to have been sent by p to q but has been created by $p?$, which may not be p, where e_i is the message body (ciphertext).
- Networks: Modeled as bags of messages.
  - Sending a message is formalized as putting it in the bag.
  - If the bag contains $\mathtt{m}i(p?, p, q, e_i)$, then q can receive it.
  - Then q believes that it originates in p, although it is not true.
  - Suppose that messages are never deleted from the bag.

Brushup (3)

- Three observable values:

```
op network : System -> Network
op rands   : System -> RandBag
op nonces  : System -> NonceBag
```

- Formalization of sending messages:

```
op sdm1 : System Principal Principal Random -> System {constr}
op sdm2 : System Principal Principal Principal
                              Random Nonce -> System {constr}
op sdm3 : System Principal Principal Principal
                               Nonce Nonce -> System {constr}
```

- Formalization of faking messages:

```
op fkm11 : System Principal Principal Message1    -> System {constr}
op fkm12 : System Principal Principal Nonce       -> System {constr}
op fkm21 : System Principal Principal Message2    -> System {constr}
op fkm22 : System Principal Principal Nonce Nonce -> System {constr}
op fkm31 : System Principal Principal Message3    -> System {constr}
op fkm32 : System Principal Principal Nonce       -> System {constr}
```

Brushup (4)

- Equations for `sdm2`:

```
ceq network(sdm2(S,Q?,P,Q,R,N))
 = m2(P,P,Q,enc2(Q,N,n(P,Q,R),P)) network(S)
 if c-sdm2(S,Q?,P,Q,R,N) .
ceq rands(sdm2(S,Q?,P,Q,R,N))
 = R rands(S) if c-sdm1(S,P,Q,R) .
ceq nonces(sdm2(S,Q?,P,Q,R,N))
 = (if Q = intruder then N n(P,Q,R) nonces(S)
                    else nonces(S) fi)
 if c-sdm2(S,Q?,P,Q,R,N) .
ceq sdm2(S,Q?,P,Q,R,N)
 = S if not c-sdm2(S,Q?,P,Q,R,N) .
```

where

```
eq c-sdm2(S,Q?,P,Q,R,N)
 = (m1(Q?,Q,P,enc1(P,N,Q)) \in network(S) and
   not(R \in rands(S))) .
```

Brushup (5)

- Equations for `fkm22`:

```
ceq network(fkm22(S,P,Q,N1,N2))
  = m2(intruder,P,Q,enc2(Q,N1,N2,P)) network(S)
  if c-fkm22(S,P,Q,N1,N2) .
  eq rands(fkm22(S,P,Q,N1,N2)) = rands(S) .
  eq nonces(fkm22(S,P,Q,N1,N2)) = nonces(S) .
ceq fkm22(S,P,Q,N1,N2)
  = S if not c-fkm22(S,P,Q,N1,N2) .
```

where

```
eq c-fkm22(S,P,Q,N1,N2)
  = N1 \in nonces(S) and N2 \in nonces(S) .
```

Brushup (6)

- Formalization of Agreement Property:

```
eq inv1(S,P,Q,Q?,R,N)
 = (not(P = intruder) and
    m1(P,P,Q,enc1(Q,n(P,Q,R),P)) \in network(S) and
    m2(Q?,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)
    implies
    m2(Q,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)) .

eq inv2(S,P,Q,P?,R,N)
 = (not(Q = intruder) and
    m2(Q,Q,P,enc2(P,N,n(Q,P,R),Q)) \in network(S) and
    m3(P?,P,Q,enc3(Q,n(Q,P,R))) \in network(S)
    implies
    m3(P,P,Q,enc3(Q,n(Q,P,R))) \in network(S)) .
```

Preparation for Verification (1)

- Module `PRED-NSLPK`: Properties to verify are declared.

```
mod* PRED-NSLPK {
  inc(NSLPK)
  op inv1 : System Principal Principal Principal
                                Random Nonce -> Bool
  op inv2 : System Principal Principal Principal
                                Random Nonce -> Bool
  …
  eq inv1(S,P,Q,Q?,R,N) = … .
  eq inv2(S,P,Q,P?,R,N) = … .
}
```

Preparation for Verification (2)

- Module `BASE-NSLPK`: Verification is done by structural induction on `System`. Fresh constants used in proof scores are declared.

```
mod* BASE-NSLPK {
  inc(PRED-NSLPK)
  ops s s' : -> System
  op r : -> Random
  op n : -> Nonce
  ops p q p? q? : -> Principal
}
```

Preparation for Verification (3)

- Module `ISTEP-NSLPK`: Basic formulas to prove in the induction case (step) and induction hypotheses are declared

An instance of the I.H.

The formula to prove

I.H.

```
mod* ISTEP-NSLPK {  inc(BASE-NSLPK)
  op istep1 : -> Bool
  op istep2 : -> Bool

  eq istep1 = inv1(s,p,q,q?,r,n)
              implies inv1(s',p,q,q?,r,n) .

  eq istep2 = inv2(s,p,q,p?,r,n)
              implies inv2(s',p,q,p?,r,n) .
  -- eq inv1(s,P,Q,Q?,R,N) = true .
  -- eq inv2(s,P,Q,P?,R,N) = true .
}
```

Proof Score Templates (1)

- The following proof score can be systematically written:

I. Base case:

```
open BASE-NSLPK
-- check
  red inv1(init,p,q,q?,r,n) .
close
```

✓ Done

II. Induction case: For each transition operator t,

```
open ISTEP-NSLPK
-- fresh constants
  op a1 : -> S1 . …
-- assumptions
-- successor state
  eq s' = t(s,a1,…) .
-- check
  red istep1 .
close
```

Fragments enclosed with open & close in proof scores are called *proof passages*.

```
eq istep1 = inv1(s,p,q,q?,r,n)
            implies
            inv1(s',p,q,q?,r,n) .
```

Proof Score Templates (2)

- If t has a non-trivial effective condition c−t, the case is split into two sub-cases based on c−t.

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . …
-- assumptions
  eq c-t(s,x1,…) = true .
-- successor state
  eq s' = t(s,x1,…) .
-- check
  red istep1 .
close
```

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . …
-- assumptions
  eq c-t(s,x1,…) = false .
-- successor state
  eq s' = t(s,x1,…) .
-- check
  red istep1 .
close
```

✓ Done

Proof Score Templates (3)

$$\texttt{eq c-}t\texttt{(S,}X_1\texttt{,...) = } \boxed{C_1\texttt{(S,}X_1\texttt{,...) and ... and } C_n\texttt{(S,}X_1\texttt{,...)}}\texttt{.}$$

$C\texttt{(S,}X_1\texttt{,...)}$

✓ $C\texttt{(S,}x_1\texttt{,...)}$ may not be derived from $\texttt{c-}t\texttt{(S,}x_1\texttt{,...) = true}$ with reriting.

✓ Moreover, each $C_i\texttt{(S,}x_1\texttt{,...)}$ may not be derived from $C\texttt{(S,}x_1\texttt{,...) = true}$ with rewriting.

✓ The left proof passage on the previous page is rewritten as this:

```
open ISTEP-NSLPK
-- fresh constants
  op x1 : -> S1 . ...
-- assumptions
  -- eq c-t(s,x1,...) = true .
  eq C1(s,x1,...) = true .
  ...
  eq Cn(s,x1,...) = true .
  --
-- successor state
  eq s' = t(s,x1,...) .
-- check
  red istep1 .
close
```

Proof Score Templates (4)

- Some equations used as assumptions in proof passages may also need to be rewritten.

✓ In the induction case for `sdm2` where c-sdm2(s,…) holds. One of the equations is as follows:

rewriting

```
eq m1(q10?,q10,p10,enc1(p10,n10,q10))
     \in network(s) = true .
```

✓ This says that `network(s)` contains `m1(…)` and then there exists a bag `NW10` such that `network(s) = m1(…) NW10`.

✓ Hence, the following equation can be used:

```
eq network(s)
     = m1(q10?,q10,p10,enc1(p10,n10,q10)) nw10 .
```

where `nw10` is a constant of `Network`.

Proof Score Templates (5)

- Assume that two sets $\boldsymbol{E}_1$, $\boldsymbol{E}_2$ of equations are equivalent in a proof passage. If each equation in $\boldsymbol{E}_2$ can be derived from $\boldsymbol{E}_1$ (together with the equations available in the proof passage) with rewriting, then $\boldsymbol{E}_1$ is preferable to $\boldsymbol{E}_2$.

- *Use preferable equations as assumptions in proof passages*!

Proof Score Templates (6)

- The proof score obtained so far is called a *proof score template*. (See file `template.mod`.)
- The initial proof score can be used to verify any (invariant) properties of the abstract model of NSLPK.

✓ *Assumption on the form of effective conditions*: Although any forms can be used, the recommended form is a conjunction of literals.
If you want to use a different form such as

$(C_1(\mathrm{S}, X_1, \ldots) \texttt{ or } C_2(\mathrm{S}, X_1, \ldots)) \texttt{ and } C_3(\mathrm{S}, X_1, \ldots)$,

then convert it into a disjunctive normal form (DNF) such as

$(C_1(\mathrm{S}, X_1, \ldots) \texttt{ and } C_3(\mathrm{S}, X_1, \ldots)) \texttt{ or } (C_2(\mathrm{S}, X_1, \ldots) \texttt{ and } C_3(\mathrm{S}, X_1, \ldots))$

and use the same number of transition operators as that of the conjuncts in the DNF such as two.

Induction Case for **`fkm21`** (1)

- Let us consider the case where `c-fkm21(s,…)` holds.

✓ CafeOBJ does not return any results.

✓ So, let us look at the formula to prove

`inv1(s',p,q,q?,r,n)`

which contains

`not(P = intruder)`

in the premise.

✓ Then, this is used to split the case into two sub-cases.

```
open ISTEP-NSLPK
-- fresh constants
  ops p10 q10 : -> Principal .
  op m20 : -> Message2 .
  op nw10 : -> Network .
-- assumptions
  -- eq c-fkm21(s,…) = true .
  eq network(s) = m20 nw10 .
  --
  eq p = intruder .
-- successor state
  eq s' = fkm21(s,p10,q10,m20) .
-- check
  red istep1 .
close
```

Induction Case for `fkm21` (2)

- In the case where "`p = intrude`", CafeOBJ returns `true`, while in the case where "`(p = intruder) = false`", it does not returns `true`.

- The difference of `s` and `s'` that affects the property: `network(s)` and `network(s')`, whose diff. is `m2(intruder,p10,q10,cipher2(m20))`.

- So, the following formula is used to split the case into two sub-cases:

A_3

```
m2(intruder,p10,q10,cipher2(m20))
= m2(q?,q,p,enc2(p,n(p,q,r),n,q))
```

Induction Case for `fkm21` (3)

- If A_3 does not hold, CafeOBJ returns `true`.
- In the case where A_3 holds, instead of one equation, the following four equations are used:

```
eq q? = intruder .
eq p10 = q .
eq q10 = p .
eq cipher2(m20) = enc2(p,n(p,q,r),n,q) .
```

- CafeOBJ does not return `true` for the case.
- We notice that if "`q = intruder`", then "`m2(q?,…)`" in the premise of `inv1(s',p,q,q?,r,n)` equals "`m2(q,…)`" in the conclusion.
- So, "`q = intruder`" is used to split the case into two sub-cases.

Induction Case for `fkm21` (4)

- If "`q = intruder`", CafeOBJ returns `true`.
  But, if "`(q = intruder) = false`", it does not.

- For the latter case, we notice that if the formula A_5

  ```
  m1(p,p,q,enc1(q,n(p,q,r),p)) \in nw10
  ```

  does not hold, the premise of `inv1(s',p,q,q?,r,n)` does not hold.

Induction Case for **`fkm21`** (5)

- If A_5 does not hold, CafeOBJ returns `true`.
  But, if A_5 holds, it does not.

- For the latter case, we also notice that if the formula

  $\boxed{\texttt{m2(q,q,p,enc2(p,n(p,q,r),n,q)) \textbackslash in nw10}}$ ← A_6

  holds, the conclusion of `inv1(s',p,q,q?,r,n)` holds.

- If A_6 holds, CafeOBJ returns `true`.
  But, if A_5 does not hold, it does not.

Induction Case for `fkm21` (6)

- The remaining case is characterized by the following equations:

```
network(s) = m20 nw10,
(p = intruder) = false,
q? = intruder,  p10 = q,  q10 = p,
cipher2(m20) = enc2(p,n(p,q,r),n,q),
(q = intruder) = false,
m1(p,p,q,enc1(q,n(p,q,r),p)) \in nw10,
m2(q,q,p,enc2(p,n(p,q,r),n,q)) \in nw10 = false
```

- We can do further case analysis, but our understanding of NSLPK tells us that there seems to be some contradiction in the set of equations.

Induction Case for `fkm21` (7)

- The assumptions say that
  - There exists a valid Init message really sent by a non-intruder `p` to a non-intruder `q`.
  - There exists a Resp message `m20` whose body (ciphertext) is valid as the reply to the Init message.
  - But, `q` has not replied to the Init message.
- These must imply that `m20` has been faked by the intruder.

Induction Case for `fkm21` (8)

- To this end, the intruder needs to get either `enc2(p,n(p,q,r),n,q)` or `n(p,q,r)`.
  - It seems impossible to get the former because `q` has not replied to the Init message.
  - It also seems impossible to get the latter because `n(p,q,r)` only appears in `enc1(q,n(p,q,r),p)`, which cannot be decrypted by the intruder.

Induction Case for **`fkm21`** (9)

- A lemma candidate:

```
                                inv4(S,P,Q,N,R,M2)
not(P = intruder) and not(Q = intruder) and
m1(P,P,Q,enc1(Q,n(P,Q,R),P)) \in network(S) and
M2 \in network(S) and
cipher2(M2) = enc2(P,n(P,Q,R),N,Q)
implies
m2(Q,Q,P,enc2(P,n(P,Q,R),N,Q)) \in network(S)
```

- `inv4(s,p,q,n,r,m20)` can be used to discharge the remaining case:

```
inv4(s,p,q,n,r,m20) implies istep1
```

On Lemma Conjecture (1)

- A systematic method of lemma conjecture:
  - Case analyses are conducted until CafeOBJ returns either `true` or `false` for each proof passage.
  - For each proof passage for which CafeOBJ returns `false`, we need to conjecture a lemma.
  - Let $e_1,\ldots, e_n$ be the set of equations used as the assumptions in such a proof passage.
  - The formula `not(`E_1 `and … and` E_n`)`, referred as a *necessary lemma* NL, obtained by substituting each (fresh) constant in "`not(`e_1 `and … and` e_n`)`" with an appropriate variable is one possible lemma candidate.

On Lemma Conjecture (2)

- It is often the case that a necessary lemma NL cannot be proved in a reasonably straightforward way.
- So, we may need to find another candidate L such that L implies NL and can be proved in a reasonably straightforward way.
  - One possible way to find L is to delete some E_i from NL, namely that `not(`E_1 `and` … `and` E_{i-1} `and` E_{i+1} `and` E_n … `and` E_n`)` may be L.
- Crème, an automatic invariant prover, basically uses this systematic method.

  Masahiro Nakano, Kazuhiro Ogata, Masaki Nakamura, Kokichi Futatsugi: Crème: an Automatic Invariant Prover of Behavioral Specifications. International Journal of Software Engineering and Knowledge Engineering 17(6): 783-804 (2007)

On Lemma Conjecture (3)

- Generally, however, we do not conduct case analyses until CafeOBJ returns either `true` or `false` for each proof passage because:
  - Too many cases (for human verifiers) are generated.
  - When you understand your target system enough well, you may find some contradiction in the set of equations used as the assumptions in a proof passage even if CafeOBJ does not return false for the proof passage, as you have seen.
- *Try to understand your target system as much as possible*; a verification attempt lets you understand your target system better partly because you need to understand it better.

Lemmas for Verification of `inv1`

- We need two more lemmas:

```
eq inv3(S,M2)
   = (M2 \in network(S)
      implies
      random(nonce1(cipher2(M2))) \in rands(S) and
      random(nonce2(cipher2(M2))) \in rands(S)) .
eq inv5(S,N)
   = (N \in nonces(S)
      implies creator(N) = intruder or
              forwhom(N) = intruder) .
```

  - The latter is what is called (*Nonce*) *Secrecy Property*.

Verification of `inv3`

- We need two lemmas:

```
eq inv8(S,M1)
   = (M1 \in network(S)
      implies
      random(nonce(cipher1(M1))) \in rands(S)) .
eq inv9(S,N)
   = (N \in nonces(S)
      implies random(N) \in rands(S)) .
```

  - But, verification of `inv8` needs `inv9`, and verification of `inv9` needs `inv3` and `inv8`.
  - Circularity?

Simultaneous Induction (1)

- Inductive invariant w.r.t. an OTS S: A state predicate p is *inductive invariant* w.r.t. S iff
  - p(init) for an arbitrary initial state init, and
  - p(s) implies p(s') for an arbitrary state s and an arbitrary successor state s' of s.
- Invariant w.r.t. an OTS S : A state predicate q is *invariant* w.r.t. S iff there exists an inductive invariant p w.r.t. S such that p implies q.
- A standard way of proving that a given state predicate q is invariant w.r.t. an OTS is to find an inductive invariant p w.r.t. S such that p implies q.

$$\frac{\exists\ p \text{ such that } p\ \texttt{implies}\ q,\quad p(\texttt{init}),\ p(\texttt{s})\ \texttt{implies}\ p(\texttt{s'})}{q \text{ is invariant w.r.t. } S}$$

Simultaneous Induction (2)

- It is often the case that p is in the form
  q `and` q_1 `and` … q_n.

Let q_0 be q.

- Suppose that p is proved by structural induction on the reachable state space.
  - Base case: All needed to do is to prove `p(init)`, but it suffices to prove each q_i`(init)` for $i = 0,1,\ldots,n$.
  - Induction case: All needed to do is, on the assumption p`(s)` for an arbitrary state `s`, to prove p`(s′)` for an arbitrary successor state `s′` of `s`, but it suffices to prove each q_i `(s′)` for $i = 0,1,\ldots,n$ on the assumption p`(s)`.

Simultaneous Induction (3)

- Consequently, the proof (fragment) of each q_i can be written separately.
- *Simultaneous induction* is a style of writing the proof of q_0 `and` q_1 `and` … q_n by induction such that the proof of each q_i is written separately.
- Some advantages:
  - Proofs can be written incrementally.

    Basically the proofs you have written do not have to be modified.
  - We can avoid making CafeOBJ reduce a long formula.

    CafeOBJ may not return any results in a reasonable time even after doing some case analyses.

Simultaneous Induction (4)

- Precisely the proof of `inv8` does not use `inv9` as a lemma (in a usual sense), but uses `inv9(s,N)` as an induction hypothesis, and the proof of `inv9` does not use `inv8` as a lemma, but uses `inv8(s,M1)` as an induction hypothesis.

- Likewise the proof of `inv1` uses `inv3(s,M2)`, `inv4(s,P,Q,N,R,M2)` and `inv5(s,N)` as induction hypotheses.

- Term "lemma" refers to state predicates such as q_1, …,q_n used to make an inductive invariant such as q_0 `and` q_1 `and` … q_n to prove that a state predicate such as q_0 is invariant.

Verification of Secrecy Property (`inv5`)

- We need two lemmas.
- It is left as an exercise to conjecture and prove them, which has something to do with the exercise of the lecture (see the final page).

Verification of `inv2`

- `inv1` is Agreement Property from the initiators' (p's) point of view, while `inv2` from the responders' (q's) point of view.
- Although `inv2` is not exactly symmetric to `inv1` w.r.t. NSLPK, they have some similarities.
- Hence, `inv2` can be proved in a similar way to prove `inv1`.
- The proof of `inv2` uses three lemmas, one of which is Secrecy Property (`inv5`).
- To complete the verification, we need one more lemma.
- The verification is left as an exercise.

Other Case Studies on Protocol Verification

- *i*KP (Internet Key Protocol) Electronic Payment Protocol
  K. Ogata, K. Futatsugi: Formal analysis of the *i*KP electronic payment protocols, 1st ISSS, LNCS 2609, Springer, pp.441-460 (2003).
- Horn-Preneel Micropayment Protocol
  K. Ogata, K. Futatsugi: Formal verification of the Horn-Preneel micropayment protocol, 4th VMCAI, LNCS 2575, Springer, pp.238-252 (2003).
- SET (Secure Electronic Transactions) Electronic Payment Protocol
  K. Ogata, K. Futatsugi: Equational Approach to Formal Verification of SET, 4th QSIC, IEEE CS Press, pp.50-59 (2004).
- NetBill Electronic Commerce Protocol
  K. Ogata, K. Futatsugi: Formal Analysis of the NetBill Electronic Commerce Protocol, 2nd ISSS, LNCS 3233, Springer, pp.45-64 (2004).
- TLS (Transaction Layer Security) Authentication Protocol
  K. Ogata, K. Futatsugi: Equational Approach to Formal Analysis of TLS, 25th ICDCS, IEEE CS Press, pp.795-804 (2005).
- Mondex Electronic Purse Protocol
  W. Kong, K. Ogata, K. Futatsugi: Algebraic Approaches to Formal Analysis of the Mondex Electronic Purse System, 6th IFM, LNCS 4591, Springer, pp.393-412 (2007).

Exercise

- Verify that the simpler abstract model of NSLPK made in the exercise of the previous lecture enjoys Secrecy Property.