language for protocol

January 30, 2018

1 A language for actors

Expressions:

$$e ::= n \mid b \mid e \oplus e.$$

Types for expressions will be required to be finite:

$$\tau ::= \mathbb{Z}_q \mid \mathsf{bool} \mid \tau \times \tau \mid \dots.$$

Let $\mathcal{I} = \{i_1, i_2, \dots\}$ be a set of *interface labels* with associated types τ_i. Let *Dist* and *Var* be supplies of labels for distributions and variables. Each distribution $D \in \mathit{Dist}$ is assigned a type τ_D of the form $\tau_1 \to \cdots \to \tau_k \to \mathcal{D}(\tau)$. (Function and distribution types do not appear elsewhere in the language.) Messages are pairs of values and interface labels.

Syntax for actors:

$$\begin{aligned}
\text{decl} &::= \epsilon \mid \text{handler};\ \text{decl} \\
\text{handler} &::= \mathsf{onInput}\ i\ v\ c \quad \text{where } i \in \mathcal{I},\ v \in \mathsf{Var} \\
c &::= \mathsf{if}\ e\ c\ c \\
&\quad \mid \mathsf{send}\ e \to i \\
&\quad \mid x \leftarrow D\ e_1\ \dots\ e_k;\ c \quad \text{where } d \in \mathit{Dist} \\
&\quad \mid x \leftarrow \mathsf{get};\ c \\
&\quad \mid \mathsf{put}\ e;\ c
\end{aligned}$$

Upon activation, actors may receive exactly one message and deliver exactly one message. Sends are guarded behind receives.

1.1 Typing

Let Γ be a typing environment, containing of variable assignments $x \mapsto \tau$ as well as a type for the state $\mathsf{St}(c) \mapsto \tau$. The main typing relation is then $\Gamma \vdash c$, stating that all expressions in c are well typed, all samplings in c are of the correct arity and type, and all stateful commands in c are well-typed.

Then, given a declaration d, write $\vdash d$ to mean that all of the commands in d are well-typed, and all commands in d share the same type for the state.

Given a message $m = (v, i)$, write $\vdash (v, i)$ to mean $\vdash v : \tau_i$.

1.2 Interface typing

We have an typing relation $\vdash d : I\ O$ on declarations, where I and O are subsets of $\mathcal{I}$.

$$\frac{}{\vdash \epsilon : \varnothing\ \varnothing} \qquad \frac{\vdash d : I\ O \quad \vdash c : O' \quad I \cap (O \cup O') = \varnothing}{\vdash\ \mathsf{onInput}\ i\ v\ c;\ d : I \cup \{i\}\ O \cup O'}$$

Above $\vdash c : O$ is the typing relation defined by

$$\frac{}{\vdash \mathsf{send}\ e \to i : \{i\}}$$

and the appropriate propogation rules.

Given a declaration d, let $\mathsf{In}(d), \mathsf{Out}(d)$ be the set of input and output messages to d (i.e., elements of *Msg* whose interface labels agree with the d). Similarly, for commands c, let $\mathsf{Out}(c)$ be the set of output interfaces of c.

1.3 Semantics

For each type τ, we have the semantic domain $[\![\tau]\!]$. Let $\mathcal{D}$ be the monad of finite probability distributions. A *distribution environment* Φ is a mapping $D \in \mathit{Dist} \to [\![\tau_D]\!]$.

We may then give commands a denotational semantics $[\![c]\!] : \Phi \to \mathsf{St}(c) \to \mathcal{D}(\mathsf{Out}(c) \times \mathsf{St}(c))$. Then, we may lift to declarations in order to obtain the semantics $[\![d]\!] : \Phi \to \mathsf{In}(d) \to \mathsf{St}(d) \to \mathcal{D}(\mathsf{Out}(d) \times \mathsf{St}(d))$.

2 Systems

Our syntax for systems is:

$$\mathrm{S} ::= \mathrm{decl} \mid \mathrm{S}\ ||\mathrm{S}.$$

We only need one combinator: $S_1||S_2$ runs S_1 and S_2 in parallel and, if they share interface labels accordingly, these interfaces get connected together. If the interfaces of S_1 and S_2 are disjoint, then they are simply run in parallel. (In constructive crypto, there is a separate operator for parallel composition: assuming that interface labels are not reused, this is redundant.)

Lift the St typing assignment by declaring that $\mathsf{St}(S_1||S_2) = \mathsf{St}(S_1) \times \mathsf{St}(S_2)$.

Interface typing:

$$\frac{\vdash S_1 : I_1\ O_1 \quad \vdash S_2 : I_2\ O_2}{\vdash S_1||S_2 : (I_1 \cup I_2) \setminus (O_1 \cup O_2)\ (O_1 \cup O_2) \setminus (I_1 \cup I_2)}$$

Lift the assignments In and Out according to the above rule. Define $\mathsf{Connect}(S_1, S_2) = (\mathsf{In}(S_1) \cap \mathsf{Out}(S_2)) \cup (\mathsf{Out}(S_1) \cap \mathsf{In}(S_2))$; $\mathsf{Connect}(S_1, S_2)$ are the message spaces for messages internal to S_1 and S_2. If S_1 and S_2 do not have any interfaces in common, $\mathsf{Connect}(S_1, S_2)$ is empty.

Systems also implicitly come with an initialization distribution $\mathsf{init}(S)$, which is a distribution $D \in \mathit{Dist}$ over $\mathsf{St}(S)$, which takes no arguments. This initialization distribution is lifted to compositions of systems in the obvious way.

System semantics: Systems are finally given the denotational semantics $[\![S]\!] : \Phi \to \mathsf{In}(S) \to \mathsf{St}(S) \to \mathcal{D}(\mathsf{Out}(S) \times \mathsf{St}(S))_\perp$. (We adjoin $\perp$ because systems may diverge.) This is defined to be $[\![S_1||S_2]\!]\phi m(s_1, s_2) := \mathsf{Run}_{S_1,S_2}\ m\ (s_1, s_2)$, where

$$\mathsf{Run}_{S_1,S_2} : \mathsf{In}(S_1S_2) \cup \mathsf{Connect}(S_1, S_2) \to \mathsf{St}(S_1S_2) \to \mathcal{D}(\mathsf{Out}(S_1S_2) \times \mathsf{St}(S_1S_2))_\perp$$

is given by:

```
1: if m ∈ In(S1) then
2:     (m', s1') ← [[s1]] φ m s1
3:     if m' ∈ Out(S1S2) then
4:         Return (m', (s1', s2))
5:     else
6:         (m ∈ Connect(S1, S2))
7:         Return Run m' (s1', s2)
8:     end if
9: else
10:    (m ∈ In(S2))
11:    (This case is symmetric)
12: end if
```

The above is written monadically: line two is implicitly using the bind operation of D. The above algorithm continues to deliver the current message until an external interface is reached.

Equivalence of systems Systems S and T are represented by their transition functions $\delta_S = \mathsf{St}(S) \to \mathsf{In} \to \mathcal{D}(\mathsf{Out} \times \mathsf{St}(S))$, and similarly for T. These transition functions are coalgebras over $\mathcal{F}(X) = \mathsf{In} \to \mathcal{D}(\mathsf{Out} \times X$. We first form the coproduct of these two coalgebras, which is a transition function over state space $\mathsf{St}(S) + \mathsf{St}(T)$ which takes S-states to S-distributions, and similarly for T.

Now, we ask whether two initial states are bisimilar. Let S be the state space for a $\mathcal{F}$-coalgebra. Then, for a state s, write $s \xrightarrow{m} \mu$ to mean $\mu := \delta_S(s, m)$.

Given an $\mathcal{F}$-coalgebra over state space S, we say that an equivalence relation $\sim$ is a *bisimulation* if for any $s \sim t$,

$$\forall m, \forall C \in S/\sim, \text{ if } s \xrightarrow{m} \mu \text{ and } t \xrightarrow{m} \eta \text{ then } \forall m', \mu(m', C) = \eta(m', C),$$

where $\mu(m', C)$ is the probability that μ outputs a pair m'', s' such that $m'' = m'$ and $s' \in C$.

Problem 1. Given a system S composed of subsystems $S_1, \ldots, S_n$, its transition function δ_S has more structure than listed above, in that it arises from a sequences of combinations of transitions $\delta_{S_1}, \ldots, \delta_{S_n}$.

Indeed, there is a notion of "small-step" protocol semantics, where each m is sent to its appropriate handler δ_{S_k}, and the corresponding output distribution is over pairs of states of S_k and messages that S_k can send. These messages are then rerouted to various parties of S until a message is produced that is addressed to someone outside of S.

Is there a suitable bisimulation notion that is appropriately modular with respect to the above structure?

Problem 2. It is likely the case that most protocols being run, while over joint state spaces of very large sizes, can be appropriately refined to protocols with relatively smaller state spaces. For example, if I am holding a 32-bit integer but only care about the first two bits, then I can refine this protocol to one where I am only holding two bits.

What is the appropriate refinement notion, given the abstract semantics of δ_S, such that it is compatible with bisimularity?

Problem 3. How does one tackle cryptographic arguments? Bisimularity only captures exact equivalence. Two options are:

1. Leave cryptographic arguments out of bisimularity, and keep it exact. Instead, have a trace semantics that allows you to securely transform protocols between cryptographic "worlds" (i.e., real encryption and symbolic ideal encryption). Then, bisimularity is carried out after these steps have been taken. (This is what Backes appears to be doing.)

2. Have a notion of approximate bisimularity which allows one to reason about cryptographic "worlds" directly in the bisimularity argument.

Problem 4. Given an answer to Problem 3, can we reason modularly about protocols by taking advantage of congruences? Given an appropriate distance pseudometric $d(\cdot, \cdot)$ on systems, could we prove that $d(S||S', T||T') \leq d(S, T) + d(S', T')$?