language for protocol

January 23, 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 $\tau_1 \to \dots \to \tau_k \to \mathcal{D}(\tau)$. (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:

$$\text{S ::= decl} \mid \text{S S.}$$

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

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

is given by:

1: **if** $m \in \mathsf{In}(S_1)$ **then**
2:     $(m', s_1') \leftarrow [\![s_1]\!]\ \phi\ m\ s_1$
3:     **if** $m' \in \mathsf{Out}(S_1 S_2)$ **then**
4:         Return $(m', (s_1', s_2))$
5:     **else**
6:         $(m \in \mathsf{Connect}(S_1, S_2))$
7:         Return $\mathsf{Run}\ m'\ (s_1', s_2)$
8:     **end if**
9: **else**
10:     $(m \in \mathsf{In}(S_2))$
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 Below is a possible notion of bisimilarity of systems. Given two systems S and T such that $\mathsf{In}(S) = \mathsf{In}(T) := \mathsf{In}$ and $\mathsf{Out}(S) = \mathsf{Out}(T) := \mathsf{Out}$, represented by their transistion functions $\delta_S = \mathsf{In} \times \mathsf{St}(S) \to \mathcal{D}(\mathsf{Out} \times \mathsf{St}(S))$, and similarly for T (fixing a particular distribution environment). Since $\mathsf{St}(S)$ is guaranteed to be finite, we may lift δ_S to operate on distributions of states, rather than single states.

Given a distribution D over pairs, let $\pi_1 D$ be the left projection of D, and for $x \in \mathsf{supp}(\pi_1 D)$, let $D_{|x}$ be the conditional distribution over the right projection of D, where the left component is required to be equal to x.

Then, for two distributions μ over $\mathsf{St}(S)$ and η over $\mathsf{St}(T)$, define

$$\mu \sim_1 \eta \text{ if } \forall m \in \mathsf{In}, \pi_1(\delta_S\ m\ \mu) \equiv \pi_1(\delta_T\ m\ \eta)$$

and

$$\mu \sim_{k+1} \eta \text{ if } \mu \sim_1 \eta \text{ and } \forall m' \in \mathsf{supp}(\pi_1(\delta_S\ m\ \mu)), (\delta_S\ m\ \mu)_{|m'} \sim_k (\delta_T\ m\ \eta)_{|m'}.$$

Since $\mu \sim_1 \eta$, both conditional distributions on the right hand side are well-defined.

Finally, let $\sim$ be $\lim_i \sim_i$. Two systems are *equivalent* if $\mathsf{init}(S) \sim \mathsf{init}(T)$.