The Spi Calculus

CSG 399 Lecture

Recall CSP

- Model system as a CSP process
- A specification is a property of traces
  - Often, can be represented as a process $Spec$
- Checking a specification: $Spec \sqsubseteq P$
  - Every trace of P is a trace of $Spec$

Abadi and Gordon's Approach

- Uses a different calculus of processes
  - Based on the π calculus
  - "Philosophical" alternative to CSP
- Offers different ways of specifying and verifying protocols
  - AG use equivalence with "obviously correct" system

The π Calculus

Let us start by defining the π calculus

- Just a calculus for reasoning about concurrent systems
- As in CSP, notion of processes, which can be put in parallel
- Processes may communicate by sending values over channels
- Channels have a scope (which process knows which channel)
- But channel names can be sent to other processes
  - Scope extrusion

Syntax - Values

First, let us define a syntax for terms that denote the values exchangeable between processes

A term M, N is one of:

- Name n
  - For channels, keys, nonces, primitive messages
- Pair (M, N)
- Variable

(AG also talk about integers and arithmetic operations)

Syntax - Processes

We use a more readable syntax introduced in later papers on the spi calculus

A process P is of the form:

- out $M\ N; Q$: send N on channel M, then behave as Q
- inp $M\ (x); Q$: receive a value on channel M, bind it to x in Q, then behave as Q
- $P \mid Q$: P and Q executing in parallel
- new $(n); Q$: create new name n in the scope of Q

Other Process Forms

- repeat Q: replicate Q
- match M is $N; Q$: proceed as Q if M and N are equal
- stop: do nothing and stops
- split M is $(x, y); Q$: split the pair M into x and y and behaves as Q

Example

$$
\begin{array}{l}
\textsf{new}\ (c);\\
\textsf{new}\ (d);\\
\textsf{new}\ (M);\\
(\textsf{out}\ c\ M; \textsf{stop}\ |\\
\ \textsf{inp}\ c\ (x); \textsf{out}\ d\ x; \textsf{stop}\ |\\
\ \textsf{inp}\ c\ (x); \textsf{stop})
\end{array}
$$

Semantics

The semantics of the π calculus is a relation $P \to Q$ that gives one possible next step of the execution of P.

Note that there can be many possible next steps

- Processes are nondeterministic

The definition is in two steps

- Define when two processes are structurally equivalent
- Define the reaction relation $P \to Q$

Reduction Relation $P > Q$

"P reduces immediately to Q"

- repeat $P > P \mid$ repeat P
- match M is $M; P > P$
- split (M, N) is $(x, y); P > P[M/x][N/y]$

$P[M/x]$: replace every free occurrence of x by M

Structural Equivalence $P \equiv Q$

"P and Q are basically the same process"

- $P \equiv P$
- $P \mid \mathsf{stop} \equiv P$
- $P \mid Q \equiv Q \mid P$
- $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
- $\mathsf{new}\ (m); \mathsf{new}\ (n); P \equiv \mathsf{new}\ (n); \mathsf{new}\ (m); P$
- $\mathsf{new}\ (n); \mathsf{stop} \equiv \mathsf{stop}$
- $\mathsf{new}\ (n); (P \mid Q) \equiv P \mid \mathsf{new}\ (n); Q$, if $n \notin fn(P)$

- If $P > Q$ then $P \equiv Q$
- If $P \equiv Q$ then $Q \equiv P$
- If $P \equiv Q$ and $Q \equiv R$ then $P \equiv R$
- If $P \equiv Q$ then $P \mid R \equiv Q \mid R$
- If $P \equiv Q$ then new $(n); P \equiv$ new $(n); Q$

Reaction Relation $P \to Q$

"P can execute and become Q"

- out $m\ N; P \mid$ inp $m\ (x); Q \to P \mid Q[N/x]$
- If $P \equiv P'$, $Q \equiv Q'$, and $P' \to Q'$, then $P \to Q$
- If $P \to P'$ then $P \mid Q \to P' \mid Q$
- If $P \to P'$ then new $(n); P \to$ new $(n); P'$

$P \to^* Q$ if $\exists P_1, \ldots, P_k$ with $P \to P_1 \to \cdots \to P_k \to Q$

The Spi Calculus - Terms

Toss in the ability to encrypt messages (shared key) and that of decrypting messages.

New term form:

- $\{M\}_N$

The Spi Calculus - Processes

New process form:

- decrypt M is $\{x\}_N;P$
- Intuitively, try to decrypt M with key N
  - If it succeeds, bind x to result and proceed with P
  - If it fails, process is stuck

Note that this embodies:

- Can only decrypt if you have the key
- There is enough redundancy to detect when decryption has succeeded

The Wide Mouthed Frog protocol

Two agents communicating without sharing a key

- A wants to send M to B
- A and B do not share keys
- A and B both share a key with a server S

$$A \longrightarrow S : \{K_{AB}\}_{K_{AS}}$$
$$S \longrightarrow B : \{K_{AB}\}_{K_{BS}}$$
$$A \longrightarrow B : \{M\}_{K_{AB}}$$

Modeling Security Protocols

Essentially like in CSP

- Write a process for each agent
- Put all the processes in parallel into a system

Then, prove something of interest about the process

Modeling WMF - Initiator

$$
\begin{aligned}
INIT(M) = \ & \mathsf{new}\ (KAB); \\
& \mathsf{out}\ net\ \{KAB\}_{KAS}; \\
& \mathsf{out}\ net\ \{M\}_{KAB}; \\
& \mathsf{stop}
\end{aligned}
$$

Assumes a channel net representing the “network”

Modeling WMF - Server

$$
\begin{aligned}
SERVER = \mathsf{repeat\ inp}\ & net\ (x); \\
& \mathsf{decrypt}\ x\ \mathsf{is}\ \{y\}_{KAS}; \\
& \mathsf{out}\ net\ \{y\}_{KBS}; \\
& \mathsf{stop}
\end{aligned}
$$

Modeling WMF - Receiver

$$\begin{aligned} RESP = {} & \mathsf{inp}\ net\ (x); \\ & \mathsf{decrypt}\ x\ \mathsf{is}\ \{y\}_{KBS}; \\ & \mathsf{inp}\ net\ (x); \\ & \mathsf{decrypt}\ x\ \mathsf{is}\ \{z\}_{y}; \\ & F(z) \end{aligned}$$

Modeling WMF - System

$$\begin{aligned} SYS(M) = {} & \text{new } (KAS); \\ & \text{new } (KBS); \\ & (INIT(M) \mid RESP \mid SERVER) \end{aligned}$$

If F does not contain free occurrences of KAS and KBS:

- $SYS(M) \rightarrow^* F(M)$
- Running the protocol can yield $F(M)$
- This is a sanity check: the protocol can make progress

Specifying Secrecy

Intuition:

- Message exchange is kept secret if the system exchanging message M is indistinguishable from the outside from the system exchanging message M'

Formally:

- Message exchanged is kept secret if for every M, M':

  If $F(M) \simeq F(M')$, then $SYS(M) \simeq SYS(M')$

Process Equivalence

We want to define a notion of what it means for two processes to be indistinguishable (called equivalent)

- There are many possible choices, depending on what one means by equivalent
- A pastime in the process calculus world is to define notions of equivalences
  - Different equivalences have different properties
  - Some are easier to establish than others
- Structural equivalence is an equivalence
  - Too fine
  - Really just a form of syntactic equivalence

Testing Equivalence

AG use testing equivalence as the notion of equivalence

Two processes are testing equivalent, written $P \simeq Q$, if they are indistinguishable to any other process

No process R can distinguish:

- If it is running in parallel with P
- If it is running in parallel with Q

Barbs

Define a predicate describing the channels on which a process can communicate

- A *barb* β is an input or an output channel, where output channels are marked by a bar $\overline{m}$

P exhibits barb β, written $P \downarrow \beta$, is defined by

- out $m\ M; P \downarrow \overline{m}$
- inp $m\ (x); P \downarrow m$
- If $P \downarrow \beta$ then $P \mid Q \downarrow \beta$
- If $P \downarrow \beta$ and $\beta \notin \{m, \overline{m}\}$, then new $(m); P \downarrow \beta$
- If $P \equiv Q$ and $Q \downarrow \beta$, then $P \downarrow \beta$

Tests

We generalize to P may eventually exhibit barb β, written $P \Downarrow \beta$, by:

- If $P \downarrow \beta$ then $P \Downarrow \beta$
- If $P \rightarrow Q$ and $Q \Downarrow \beta$, then $P \Downarrow \beta$

A test is a closed process R and a barb β—think, process R trying to see if the tested process can be made to exhibit barb β

$P \sqsubseteq Q$ if for all (R, β), $(P \mid R) \Downarrow \beta$, then $(Q \mid R) \Downarrow \beta$

$P \simeq Q$ if $P \sqsubseteq Q$ and $Q \sqsubseteq P$

Testing Equivalence is a Congruence

One can check that testing equivalence has a nice property:

- If P and Q cannot be distinguished by a third process R in parallel, it turns out that P and Q can be used interchangeably in any context

Formally:

- $\simeq$ is a congruence
- If $P \simeq Q$, then $C[P] \simeq C[Q]$, when $C[\cdot]$ is a closed context—a closed process with a hole

Specifying Authentication

Intuition:

- The system where message M is exchanged using the protocol is indistinguishable from a system where message M “magically” makes it to the responder.

The “Specification” System

$$
\begin{aligned}
RESP'(M) = {} & \mathsf{inp}\ net\ (x); \\
& \mathsf{decrypt}\ x\ \mathsf{is}\ \{y\}_{KBS}; \\
& \mathsf{inp}\ net\ (x); \\
& \mathsf{decrypt}\ x\ \mathsf{is}\ \{z\}_{y}; \\
& F(M)
\end{aligned}
$$

$$
\begin{aligned}
SYS'(M) = {} & \mathsf{new}\ (KAS); \\
& \mathsf{new}\ (KBS); \\
& (INIT(M) \mid RESP'(M) \mid SERVER)
\end{aligned}
$$

Formalizing Authentication

- Message is authenticated if for all M:

$$SYS(M) \simeq SYS'(M)$$

Where's the Adversary?

It is implicit in the model!

- All properties expressed as $P \simeq Q$
- $P \simeq Q$ means no third process (the adversary) can make it so that something can be distinguished between P and Q
- Third process can intercept messages, decrypt them if he knows the key, take messages apart, send new messages, etc
- Thus, the third process can be thought of as an instance of a Dolev-Yao adversary

Final Notes

How do you check $P \simeq Q$?

- Prove it explicitly by applying definitions
- Develop a proof system for $\simeq$
- Define an equivalence that is easier to establish, that implies $\simeq$

Alternative:

- Keep spi calculus as a language
- Use different specification and verification techniques
  - Proverif (uses logic programming)
  - Correspondence assertions (uses a type system)