Lecture 13: Roadmap

- From the Infinite to the Finite
- Proving the Refinement Relation
- The AUY Model
- Asynchronous Req/Del Case
- Implementing One Layer on Another
- The Protocol
From the Infinite to the Finite

Assume environment cannot reorder messages. Then we have the following protocol, where $\oplus$ is used for modulo 2 addition:

**Sender’s Protocol:**

\[
i := 0; \ z := \lambda; \ \text{read} \ \ y; \\
\text{do forever} \\
\quad \text{if} \ (z \neq i \oplus 1) \quad \text{in}(m) \\
\quad \quad \text{then send} \ \langle i, y \rangle \quad \text{Effect:} \\
\quad \quad \text{else} \ i := i \oplus 1; \ \text{read} \ y \quad z := m \\
\text{end;}
\]

**Receiver’s Protocol:**

\[
j := 0; \ z', \ x := \lambda \\
\text{do forever} \\
\quad \text{if} \ z' \neq j \quad \text{in}(m) \\
\quad \quad \text{then send} \ j; \quad \text{Effect:} \\
\quad \quad \text{else} \ \text{write} \ x; \ j := j \oplus 1 \quad (z', \ x) := (\pi_1(m), \pi_2(m)) \\
\text{end;}
\]
Proof of Correctness

The proof is by showing that the finite-state protocol $A^{fs}$ is an implementation of the infinite-state protocol $A^{st}$. This is done by defining a mapping $\Psi$ between runs for $A^{fs}$ and runs $A^{st}$ that preserves reading and writing of data elements, and preserves fairness. That is, for every run $r$ of $A^{fs}$, we require:

- $X^{\Psi(r)} = X^{r}$;
- $c^{\Psi(r)}(t) = c^{r}(t)$ for all time $t$, where $c$ is the input counter;
- $Y^{\Psi(r)}(t) = Y^{r}(t)$ for all time $t$;
- If $r$ is fair then so is $\Psi(r)$.
Defining $\Psi$

- $y^{\Psi}(r)(t) = y^r(t)$;
- $i^{\Psi}(r)(t) = c^r(t)$;
- $j^{\Psi}(r)(t) = |Y^r(t)|$;

$$z^{\Psi}(r)(t) = \begin{cases} 
  i^{\Psi}(r)(t) & \text{if } z^r(t) = i^r(t) \\
  i^{\Psi}(r)(t) + 1 & \text{if } z^r(t) = i^r(t) \oplus 1 \\
  z^r(t) & \text{otherwise}
\end{cases}$$

- For the sequence of sent $S$-messages we have:

$$b^S_{\Psi}(r)(t) = \begin{cases} 
  \langle \rangle & \text{if } t = 0 \\
  b^S_{\Psi}(r)(t-1) & \text{if } t > 0, \quad b^r_S(t) = b^r_S(t-1) \\
  b^r_{S(t-1)} \cdot \langle i^{r(t-1)}, y^{r(t-1)} \rangle & \text{otherwise}
\end{cases}$$
Proving the Refinement Relation

It is easy to see that $\Psi(r)(0)$ is an initial state of some run of $A^{st}$. We now have to show (by induction) that this is true for every time $t > 0$.

The proof is based on the following observation:

**Lemma** For every run $r$ of $A^{fs}$, for every $t > 0$:

- $(c^{r(t)} \equiv i^{r(t)}) \mod 2$ and $(j^{r(t)} = |Y^{r(t)}|) \mod 2$;

- If $c^{r(t)} = z^{r(t)} = 0$ then there exists some $t' < t$ such that
  $$|Y^{r(t')}| = 0 \quad \text{and} \quad b_{R}^{r(t'+1)} \neq b_{R}^{r(t')};$$

- If $|Y^{r(t)}| = \pi_1(z^{r(t)}) = 0$ then there exists some $t' < t$ such that
  $$|c^{r(t')}| = 0 \quad \text{and} \quad b_{S}^{r(t'+1)} \neq b_{S}^{r(t')};$$

- $|Y^{r(t)}| \leq c^{r(t)} \leq |Y^{r(t)}| + 1$;
Continuation of Lemma

- If \( c^r(t) = k > 0 \) then there exists some \( t_1 < \ldots t_k < t \) such that for all \( \ell = 1, \ldots, k \)
  \[ |Y^r(t_\ell)| = \ell \text{ and } b^{r(t_{\ell+1})}_R \neq b^{r(t_\ell)}_R \]
  (so that a message was sent by \( R \) at the point \( r(m_\ell) \).

- In addition, if \( z^r(t) = k + 1 \), then \ldots (with \( k + 1 \) replacing \( k \).)

- If \( |Y^r(t)| = k > 0 \) then there exists some \( t_1 < \ldots t_{k-1} < t \) such that for all \( \ell = 0, \ldots, k - 1 \)
  \[ |c^{r(t_\ell)}| = \ell \text{ and } b^{r(t_{\ell+1})}_S \neq b^{r(t_\ell)}_S \]
  (so that a message was sent by \( S \) at the point \( r(m_\ell) \).

- In addition, if \( \pi_1(z^r(t)) = k + 1 \mod 2 \), then \ldots with \( k \) replacing \( k - 1 \).
The AUY Model

Message transmission is in synchronous clocked rounds: Each step (or round) consists of three phases: send, receive, and local. The symbols transmitted on the channel are 0, 1, and $\lambda$ (denoting an empty message.) Possible errors:

- **Deletion**: 0/1 is sent, $\lambda$ is received.
- **Mutation**: 0/1 is sent, 1/0 is received.
- **Insertion**: $\lambda$ is sent, 0/1 is received.

If all error types are present, every sequence of length $n$ over \{0, 1, $\lambda$\} can be altered to any other sequence of same length over same alphabet. Thus, $R$ can gain no information from what it receives.

Note that $A^f$ does not solve the problem, since the AUY model doesn’t allow for messages of the type $\langle y, i \rangle$. We therefore have to encode messages using only such as $\langle y, i \rangle$ using \{0, 1, $\lambda$\}. 

_Distributed Computing, NYU, Spring 2002_
AUY with Insertion/Deletion

To send \langle y, i \rangle, S can send first \( i \), then, on the next round, send \( y \). Thus, each iteration of the loop consists of two steps (and two send events.) \( S \) should therefore have two variable to store received messages, say \( z_1 \) and \( z_2 \). The receive instructions can be put in the protocol (because of synchronicity.) The protocol is therefore:

**Sender’s Protocol**

\[
i := 0; \; z_1, z_2 := \lambda; \; \text{read} \; y;
\]

**do forever**

\[
\begin{align*}
& \text{send } i; \; \text{receive } z_1; \\
& \text{send } y; \; \text{receive } z_2;
\end{align*}
\]

**if \( z_1 = i \oplus 1 \)**

\[
\begin{align*}
& \text{then } i := i \oplus 1; \; \text{read } y
\end{align*}
\]

**end;**

**Receiver’s Protocol**

\[
j := 0; \; z'_1, z'_2 := \lambda;
\]

**do forever**

\[
\begin{align*}
& \text{send } j; \; \text{receive } z'_1; \\
& \text{send } \lambda; \; \text{receive } z'_2;
\end{align*}
\]

**if \( \langle z'_1, z'_2 \rangle = \langle j, \lambda \rangle \)**

\[
\begin{align*}
& \text{then write } z'_2; \; j := j \oplus 1
\end{align*}
\]

**end;**
Proof of Correctness

Define a refinement mapping between runs of $A^{di}$ to runs of $A^{fs}$. Note that each two rounds of $A^{di}$ map to a single step of $A^{fs}$. If both $i$ and $y$ are delivered uncorrupted (\(=\) undeleted) to $R$, then we say that $\langle i, y \rangle$ is delivered uncorrupted. Otherwise, we say that $\langle i, y \rangle$ is deleted.

We define uncorrupted/deleted messages of $R$ similarly. (Note that we don’t have to worry about insertions for the second phase messages, but we do anyway.)

A run of $A^{di}$ is fair if infinitely many 2-blocks messages are being delivered uncorrupted to both parties.

The rest of the proof is straightforward.
Other Error types

Obviously, $A^{di}$ cannot deal with mutation errors. However, for channels that mutate (but do not both delete and insert) we can solve the problem by defining other encoding, as long as they preserve the following for every message $m \in \{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle, 0, 1 \}$:

- **Unique decodability.** If $e(m)$ is received uncorrupted, then the recipient knows that it is uncorrupted, and that it is an encoding of $m$.

- **Corruption Detectability.** If $e(m)$ is corrupted, then the recipient knows it is corrupted.

For the case of $A^{di}$, we used:

$$e^{di}(\langle y, i \rangle) = \langle i, y \rangle \text{ and } e^{di}(i) = \langle i, \lambda \rangle$$

for every $i, y \in \{0, 1\}$. This obviously satisfies the conditions for the deletion/insertion case.
Other Encodings

For the deletion/mutation case, we can have:

\[
es_{dm}(\langle 0, 0 \rangle) = \langle 1, \lambda, \lambda, \lambda \rangle
\]
\[
es_{dm}(\langle 0, 1 \rangle) = \langle \lambda, 1, \lambda, \lambda \rangle
\]
\[
es_{dm}(\langle 1, 0 \rangle) = \langle \lambda, \lambda, 1, \lambda \rangle
\]
\[
es_{dm}(\langle 1, 1 \rangle) = \langle \lambda, \lambda, \lambda, 1 \rangle
\]
\[
es_{dm}(0) = es_{dm}(\langle 0, 0 \rangle)
\]
\[
es_{dm}(1) = es_{dm}(\langle 0, 1 \rangle)
\]

The properties are obviously met. In fact, mutations can be easily corrected, and deletion can be detected when an all-empty 4-block is received.
And Finally . . .

For the mutation/insertion case, we just take the encoding for the deletion/mutation case, and reverse the roles of $\lambda$ and $1$. E.g.,

$\epsilon_{mi}(\langle 0, 0 \rangle) = \langle \lambda, 1, 1, 1 \rangle$

**Fairness** for both mutation cases is defined relative to 4-blocks.

This allows us to define the required refinement mapping that maps every run of the AUY protocols to a run of $A^f$, and maps fair runs to fair runs. We can therefore conclude that all the AUY protocols solve the Sequence Transmission Problem.
Asynchronous Reo/Del Case

A directionless layer $L$ is a behavior over SNDs and RCVs over some alphabet $M$. Two layers are disjoint if they operate on disjoint message alphabets. Layer $L_1$ refines layer $L_2$, denoted by $L_1 \subseteq L_2$, if $L_1$’s alphabet contain $L_2$’s, and the projection of every $L_1$-behavior onto $L_2$’s actions is a $L_2$-behavior. Disjoint layers can be composed in the obvious way, resulting in a new layer over their joint message alphabets.

A communication layer $L$ between $s$ and $r$ is a directionless layer together with a partition of the message alphabet to $M_L^{sr}$ and $M_L^{rs}$:

If either $M_L^{rs}$ or $M_L^{sr}$ then the layer is one-way.
Properties for Communication Layers

Let $\alpha$ be a sequence over SND and RCV elements over some message alphabet $\mathcal{M}$. Consider a function $\text{CAUSE}$ that maps RCV events in $\alpha$ to earlier SND events in $\alpha$.

**No Corruption** for every $m$, for every $\text{RCV}(m)$-event $\pi$, $\text{CAUSE}(\pi)$ is a $\text{SND}(m)$-event.

**No Duplication** $\text{CAUSE}$ is one-to-one.

**No Loss** $\text{CAUSE}$ is onto.

**No Reordering** for every RCV-events $\pi_1$ and $\pi_2$, if $\text{CAUSE}(\pi_1) \rightarrow \text{CAUSE}(\pi_2)$ then $\pi_1 \rightarrow \pi_2$.

**Progress** for every $m$, if there are infinitely many $\text{SND}(m)$-events, then $\text{CAUSE}$ has infinitely many $\text{SND}(m)$-events.

**Weak Progress** if there are infinitely many $\text{SND}(\mathcal{M})$-events, then $\text{CAUSE}$ has an infinite range.
Layer Families

FIFO layers are those that are consistent with all properties but weak progress.

Order Preserving (OP) layers are those that are consistent with no corruption, no reordering, and weak progress. Thus, traces in this class are those in which every sent messages can be received 0 or more times, but before any later sent message is received.

Non Duplicating (ND) layers are those that are consistent with no corruption, no duplication, and progress. That is, ever message is received at most once, and if infinitely many copies of it are sent then infinitely many are received.
Properties of ND Layers

- closed under message restriction. I.e., if a ND layer is projected onto a smaller message alphabet, the result is an ND layer.

- closed under composition (with disjoint layers)

- any message partition defines a layer decomposition

- any prefix of an ND-consistent behavior is ND-consistent

- any sub-multiset of pending ND-messages after a finite ND-trace can be delivered at any time after the trace

- after any finite period of activity, an ND-layer may act just like a ND-layer starting from the start state.
Implementing One Layer on Another

To implement a **FIFO** layer over a **ND** layer, we will proceed in two stages.

- Implement an **OP** layer over a **ND** Layer, and then
- Implement a **FIFO** layer over a **OP** layer.

**BUT** we already know how to implement a **FIFO** layer over a **OP** layer.
1-way OP-Layer over a ND-Layer

**Goal:** Find $A^s$ and $A^t$ to implement:

$S$ sends messages to $R$ only upon explicit requests on the form of *query* messages, and then it sends the most recent incoming OP-message. It saves in *unanswered* the number of unanswered queries.

$R$ always sends queries, keeping in *pending* the number of unanswered queries. For each $m \in M$, $R$ keeps in $count[m]$ the number of copies of $m$ it received since its last OP-receive action. When it performs an OP-receive, it sets *old* to be *pending*, so when it later observes $count[m] > old$, it can safely conclude that $m$ was sent to it after its last OP-receive.
## The Protocol

### $A^s$

- recent := $\lambda$
- unanswered := 0

### $SND_{op}(m)$

**effect:**
- recent := $m$

### $RCV(query)$

**effect:**
- unanswered++

### $SND_{nd}(m)$

**precondition:**
- unanswered > 0
- $m = recent \neq \lambda$

**effect:**
- unanswered--

### $A^r$

- pending, old := 0
- count array$[M]$ init 0

### $RCV(m)$

**precondition:**
- count$[m] > old$

**effect:**
- count$[w] := 0$ for all $w \in M$
- old := pending

### $SND(query)$

**effect:** pending++

### $RCV_{nd}(m)$

**effect:**
- pending--
- count$[m]++$