Time, liveness & safety

Last class

- What, how & why.

- The asynchronous model
  - No bounds on message delays (time b/w send & receive)
  - No bounds on processing delay
  - No global clocks

- Network & fairness

- Message passing

Today: How to reason about protocols (algorithms) designed for the asynchronous model.

Reasoning about programs
- Is it correct?
- What outputs can it produce?
- How long does it take to execute some ops?
  How many messages does it send (...)?
- Is it fault tolerant?
- What consistency guarantees does it provide?

```plaintext
int i = 0;       // 1
while (true) {   // 2
  if (i % 7 == 0 && i % 2 == 0) { // 3
    print (i); // 4
  }
  i++;         // 5
}
```

(a) print only odd numbers

(b) prints in ascending order

\[ \text{Swap } (x,y) \]
\[ x = x \land y \]
\[ y = x \land y \]
\[ x = x \land y \]

(a) \( x_{\text{out}} = y_{\text{in}} \); \( y_{\text{out}} = y_{\text{in}} \)

**Observation:** Correctness often depends on order of operations
Today

Q1 Modeling Order Of Events In Distributed Systems: Traces

Q2 Correctness In Terms Of Trace Properties
  / Safety \ Liveness

Q3 Recovering Traces From Actual Executions

But First: Need To Add Something To Model

I/O Automaton
  (Nancy Lynch & Others)

\[ e_0 \; e_1 \; e_2 \]
ORDER OF EVENTS ON ONE PROCESS

\[ e_5 \leq e_6 \]

\[ \text{Partial Order} \]

\[ \text{Causality} \]

\[ \mathcal{S}(I_1, e_1); (I_2, e_2) \cdots \\
(I_2, e_5); (e_1, e_5) \\
(e_5, e_3); (e_1, e_6) \\
(e_6, e_2) \]
Total Order

Trace

\[ \mathbb{N}^+ \rightarrow \text{event} \]

Correctness as a Trace Property

Correctness conditions restrict what traces can be produced by the program

(a) Prints in Ascending Order

\[ T_0 \quad 0 \quad 7 \quad 21 \quad \ldots \]

\[ T_1 \quad 0 \quad 7 \quad 14 \quad 21 \quad \ldots \]

\[ T_2 \quad 0 \quad 14 \quad 7 \quad 21 \quad \ldots \]
No messages are received without being sent first.

Why look at connectness this way?

- Concurrent processes
- Often visible to the rest of the world as one system
- Correctness is all about what
Correctness Invariants

\textbf{Safety} Condition Should Always Hold.
- No new before send.
- No even number is pointed

\textbf{Liveness} Condition Will Eventually Hold
- Puts to S3 will eventually update val
  \[ \text{get("grade/alice")} \rightarrow \text{"I"} \]
  \[ \text{Put("grade/alice", "A")} \]
  A message sent infinitely often will be received infinitely often
Test: Given a trace $T$ can you find an extension (suffix) $S$, s.t.

$TS$ is admissible

+ meets liveness condition

Reading: Liveness, uniform liveness, absolute liveness

- Liveness. Given $T$, $\exists S$ s.t.

$TS$ is admissible & the liveness condition holds

- Uniform $\exists S$, s.t. $\forall T$, if $T$ is admissible $TS$ is admissible & liveness condition holds
- Absolute: Given a trace $T$ that is admissible for which the LIVENESS condition holds, $T'$ an admissible trace

$$T'$$ is admissible + liveness holds.

Clocks:

**RECOVERING TRACES IN REALITY**

![Diagram of a net with processes $P_1$, $P_2$, and $P_3$.]

Asynchrony

Assumptions:
- No global clock
- Processes have local clocks but
  (a) Run at unknown freq.
  (b) Freq. differs btw processes

What is desired: Capture Causality

- Lampert's approach
- Vector clocks

Where Chandy–Lamport fits in?

assert

int x
recv x
ret
Channels are shaded.

$P_0 ightarrow P_2$

$P_{1} \rightarrow P_2$