Models

- What is a model?
  - An abstraction of the relevant properties of a system

- Why construct or learn a model?
  - Real world is complex, a model makes assumptions and simplifies
  - Helps us tackle the complexities
  - Often a symbolic model with properties expressed in mathematical symbols and relationships
Modeling

- Very important skill
  - Useful when *solving* problems (e.g. making an algorithm)
  - When *predicting* behavior (e.g. cost in number of messages)
  - When *evaluating* and *verifying* a solution (e.g. simulation)
Modeling

Different types of models:

- Continuous models
  - Often described by differential equations involving variables which can take real (continuous) values

- Discrete event models
  - Often described by state transition systems: system evolves, moving from one state to another at discrete time steps

This course: a model of distributed computing (discrete)
Model of distributed computing

- Biggest challenge when modeling is choosing the right level of abstraction!

- The model should be powerful enough to construct impossibility proofs
  - A statement about all possible algorithms in a system

- Our model should therefore be:
  - **Precise**: explain all relevant properties
  - **Compact**: explain concisely a class of distributed systems
Sets contain elements, which can be sets:
- \( A = \{ a, \{ d \}, \{ a \} \} \) (Set \( A \) contains \( a \), and the sets \( \{ d \} \) and \( \{ a \} \))

Sometimes we do not want to list all the members of a set, then we can write:
- \( \{ x \mid x \text{ has some property} \} \), e.g. \( \{ x \mid x \text{ is an even number} \} \), the set of all even numbers

Order not important, number of occurrences not important:
- \( \{ a, b \} = \{ a, a, b \} = \{ b, a \} \)
- \( \{ a, \{ a \} \} \neq \{ a \} \)

Exception: multi-sets, number of occurrences are important:
- \( \{ a, b \} \neq \{ a, a, b \} \)

If an element \( x \) belongs to a set \( S \), we write \( x \in S \)

Every sets contains the empty set, \( \emptyset \). I.e. for all sets, the following is true: \( \emptyset \in S \)
Operations on sets, orders

- **Cartesian product** $\times$:
  - $X \times Y$, the set $\{ab \mid a \in X \text{ and } b \in Y\}$
  - Example, $X = \{a, b, c\}$ $Y = \{c, d, e\}$
    - $X \times Y = \{ac, ad, ae, bc, bd, be, cc, cd, ce\}$

- **Union** $\cup$:
  - $X \cup Y$ is the set $\{a \mid a \in X \text{ or } a \in Y\}$
  - Example: $X = \{a, b\}$ $Y = \{a, d, e\}$
  - $X \cup Y = \{a, b, d, e\}$

- Sets are ordered by the subset, $\subseteq$, relationship:
  - $X \subseteq Y$ iff $a \in X$ then $a \in Y$ for all $a \in X$
    - $\{a, b\} \subseteq \{a, b, c\}$
    - $\{a, b, c\} \subseteq \{a, b, c\}$
    - $\{a, d\} \nsubseteq \{a, b, c\}$
Informal model of a distributed system

- A distributed system consists of
  - a bunch of processors
  - connected by a network
  - which communicate by message passing

- How do we make a useful abstract model of this?
Formal model

- We formalize the whole distributed system as a *state transition system* (STS),

- Commonly used to model discrete systems

- We also formalize each process/node as a STS

- They are commonly used to describe systems, algorithms, and software
State Transition System (informal)

- A state transition system consists of
  - a bunch of states
  - rules describing for each state what other states it can go to (transition relation)
  - a subset of the states which the system can start in (initial states)
Example algorithm:

X := 0;
while (X < 3) do
    X = X + 1;
endwhile
X := 1

Formally:

- States \{X0, X1, X2\}
- Possible transitions \{X0\rightarrow X1, X1\rightarrow X2, X2\rightarrow X1\}
- Start states \{X0\}
State transition system - formally

A STS is formally described as the triple:
\[(C, \rightarrow, I)\]

Such that:
1. \(C\) is a set of states
2. \(\rightarrow\) is a subset of \(C \times C\), describing the possible transitions \((\rightarrow \subseteq C \times C)\)
3. \(I\) is a subset of \(C\) describing the initial states \((I \subseteq C)\)

Note that the system may have several transitions from one state
E.g. \(\rightarrow = \{X2 \rightarrow X1, X2 \rightarrow X0\}\)
We will use an STS to model 2 different things:
- The state of the whole distributed system
- The computation on each local processor

Two make it clear, we have different names for the different parts of the two STSs
- For the STS of the whole distributed system:
  - The states are called \textit{configurations}
  - The state transitions are called \textit{configuration transitions}

- For the STS of every local process:
  - The states are called \textit{states}
  - The state transitions are called \textit{events}
Distributed System as a STS

- We will model the whole distributed system as a STS \((C, \rightarrow, I)\).

- The state of the whole distributed system, \(C\), can be described by:
  - the current configuration of each processor in the system
  - the messages in transit in the network
Executions in the model

- A configuration $\gamma$ is *terminal* if there exist no transition from $\gamma$ to any other configuration $\delta$.

- An *execution* in the distributed system is a sequence of configurations $(\gamma_1, \gamma_2, \gamma_3, \ldots)$ such that:
  - $\gamma_1$ is an initial configuration, i.e. $\gamma_1 \in I$.
  - there is a transition between $\gamma_i \rightarrow \gamma_{i+1}$ for every $i \geq 1$.
  - the size of the sequence is either infinite or the last configuration is terminal.

- A configuration $\gamma$ is *reachable from* configuration $\delta$ if there exists a sequence $\delta = \gamma_1, \gamma_2, \ldots, \gamma_k = \gamma$, such that $\gamma_i \rightarrow \gamma_{i+1}$ for all $1 \leq i < k$.

- A configuration $\gamma$ is *reachable* if it is reachable from an initial configuration $\delta$. 
Transitions

What types of transitions are allowed? (→)

- Answer: depends on the type of distributed system

We will model two different types of distributed systems, those using:

- Synchronous message passing
- Asynchronous message passing
Types of message passing

- Synchronous message passing:
  - The sending and receiving of a message \( m \), happens during one configuration transition

- Asynchronous message passing
  - The sending of a message \( m \), and the receipt of \( m \) occur during two different configuration transitions (not necessarily consecutively)

Asynchronous message passing is more general:
Synchronous message passing is a special case of asynchronous message passing
Transitions

- What exactly happens during a transition,
  - Answer: again, depends on the type of distributed system

Let's back off a moment, and look at how individual processes compute!
Asynchronous Message Passing

- Each processor in the distributed system runs a local algorithm, which either:
  - Sends a message
  - Receives a message
  - Updates its local variables (state)
Local Algorithms

- The local algorithm will also be modeled as a STS, and in each configuration transition one of the following three events happen:
  - A processor changes state from one state to another state (internal event)
  - A processor changes state from one state to another state, and sends a message to the network destined to another processor (send event)
  - A processor receives a message destined to it and changes state from one state to another (receive event)
Local Algorithms as STS

- Let $M$ be the set of all possible messages that can be sent (each one destined to a particular process).

- A local algorithm on a processor is modeled by the STS $(Z, I, \mathcal{I}, \mathcal{S}, \mathcal{R})$ where:
  - $Z$ is a set of all the states of the processor.
  - $I$ is a subset of $Z$, containing the initial states of the STS.
  - $\mathcal{R}$ is a subset of $Z \times M \times Z$, describing the possible receive events.
  - $\mathcal{S}$ is a subset of $Z \times M \times Z$, describing the possible send events.
  - $\mathcal{I}$ is a subset of $Z \times Z$, describing the possible internal events.
Example of a local algorithm

- Lets do a local algorithm running on processor $p$, which can receive a ping message, sends back a pong and increase a local counter (up to 2 messages)

```
        ping  pong
   ________      ________
  /          \
 `----------'
     p

     count++
```
Ping Pong local algorithm

- $M = \{\text{ping, pong}\}$
- $Z = \{c0, c0r, c0s, c1, c1r, c1s, c2\}$
- $I = \{c0\}$
- $R = \{(c0, \text{ping}, c0r), (c1, \text{ping}, c1r)\}$
- $S = \{(c0r, \text{pong}, c0s), (c1r, \text{pong}, c1s)\}$
- $I = \{(c0s, c1), (c1s, c2)\}$
Ping Pong

Ping Pong STS

- \( M = \{ \text{ping, pong} \} \)
- \( Z = \{ c0, c0r, c0s, c1, c1r, c1s, c2 \} \)
- \( I = \{ c0 \} \)
- \( R = \{ (c0, \text{ping}, c0r), (c1, \text{ping}, c1r) \} \)
- \( S = \{ (c0r, \text{pong}, c0s), (c1r, \text{pong}, c1s) \} \)
- \( I = \{ (c0s, c1), (c1s, c2) \} \)

Graphically we have the following:

![Diagram of ping pong game states and transitions](image-url)
Model of the distributed system

- Lets go back to the STS of our distributed system

- Based on the STS of a local algorithm, we can now define for the whole distributed system:
  - Its configurations
    - We want it to be the state of all processes and the network
  - Its initial configurations
    - We want it to be all possible configurations where every local algorithm is in its start state and an empty network
  - Its transitions, and
    - We want each local algorithm state event (send, receive, internal) be a configuration transition in the distributed system
A distributed system running on processes \( \{p_1, \ldots, p_n\} \), where each \( p_i \) is running a local algorithm \((Z_{pi}, I_{pi}, I_{pi}, S_{pi}, R_{pi})\), is modeled by an STS \((C, \rightarrow, I)\).

Where

- \( C = (c_{p1}, c_{p2}, \ldots, c_{pn}, M) \), where \( c_{pi} \in Z_{pi} \) and \( M \) is a multiset of messages.

I.e., a configuration in the distributed system is a sequence of every processor’s current state, and a multiset of all the messages currently in transit.
Initial configurations of the distributed system

- A distributed system running on processes \( \{p_1, ..., p_n\} \), where each \( p_i \) is running a local algorithm
  
  \[
  ( Z_{pi}, I_{pi}, \| I_{pi}, \| S_{pi}, \| R_{pi} ),
  \]

  is modeled by an STS
  
  \[
  (C, \rightarrow, I)
  \]

- Where
  
  - \( C = (c_{p1}, c_{p2}, ..., c_{pn}, M) \), where \( c_{pi} \in Z_{pi} \) and \( M \) is a multiset of messages
  - \( I = (i_{p1}, i_{p2}, ..., i_{pn}, M) \), where \( i_{pi} \in I_{pi} \) and \( M \) is empty \( \emptyset \)

- I.e., the initial configuration of the distributed system can only be a configuration where every processor is in an initial state, and there are not messages in transit in the network
Transitions of the distributed system

- A distributed system running on processes \{p_1, \ldots, p_n\}, where each \( p_i \) is running a local algorithm
  \((Z_{pi}, I_{pi}, I_{pi}, S_{pi}, R_{pi})\), is modeled by an STS
  \((C, \rightarrow, I)\)

- Where
  - The transitions of the system are \( \rightarrow = T_1 \cup T_2 \cup \ldots \cup T_n \)
  - Where for all \( 1 \leq i \leq n \), \( T_i \) is the set of all configuration transitions
    - \( C_1 \rightarrow C_2 \) such that
    - \( C_1 = (p_1, \ldots, p_i, \ldots, p_n, M_1) \)
    - \( C_2 = (p_1, \ldots, p_j, \ldots, p_n, M_2) \)
    - Where one the following is true
      - \( (c, d) \in I_{pi} \) such that \( M_1 = M_2 \) and \( p_i = c \) and \( p_j = d \)
      - \( (c, m, d) \in S_{pi} \) such that \( M_2 = M_1 \cup \{m\} \) and \( p_i = c \) and \( p_j = d \)
      - \( (c, m, d) \in R_{pi} \) such that \( M_1 = M_2 \cup \{m\} \) and \( p_i = c \) and \( p_j = d \)
Constraints

- We want to make sure that when the distributed system makes transitions, the following is preserved:
  - A process $p$ can only receive a message after it has been sent to it by some other process
  - A process $p$ can only change from a state $c$ to another state $d$ if it is currently in state $c$

- Therefore, events can only happen if they are applicable…
Applicable internal events

- Any internal event \( e = (c, d) \in I_{p_i} \) is said to be applicable in an configuration \( C = (c_{p1}, \ldots, c_{p_i}, \ldots, c_{pn}, M) \) if \( c_{p_i} = c \)
- If event \( e \) is applied, we get \( e(C) = (c_{p1}, \ldots, d, \ldots, c_{pn}, M) \)

- Example, if there are 2 processors, \( p1, p2 \), and
  - Set of states on \( p1 \) and \( p2 \) are \( Z1 = Z2 = \{ s1, s2, s3 \} \)
  - \( I_{p1} = \{ (s1,s3), (s3,s2), (s2, s3) \} \)
  - \( I_{p2} = \{ (s1,s2), (s2,s3), (s3, s1) \} \)

  - The internal event \( (s2, s3) \in I_{p2} \) on \( p2 \) is applicable in all the following configurations:
    - \( (s1, s2, M), (s2, s2, M), \) and \( (s3, s2, M) \), for any multiset of messages \( M \)
Applicable send events

- Any send event \( e = (c, m, d) \in \nabla_{pi} \) is said to be applicable in an configuration \( C = (c_{p1}, \ldots, c_{pi}, \ldots, c_{pn}, M) \) if \( c_{pi} = c \)

- If event \( e \) is applied, we get \( e(C) = (c_{p1}, \ldots, d, \ldots, c_{pn}, M \cup \{m\}) \)

- Example, if there are 2 processors, \( p1, p2 \), and
  - Set of states on \( p1 \) and \( p2 \) are \( Z1 = Z2 = \{s1, s2, s3\} \)
  - \( \nabla_{p1} = \{ (s1, m, s3), (s3, n, s2), (s2, o, s3) \} \)
  - \( \nabla_{p2} = \{ (s1, n, s2), (s2, m, s3), (s3, o, s1) \} \)

  - The send event \( (s2, m, s3) \in \nabla_{p2} \) on \( p2 \) is applicable in all the following configurations:
    - \( (s1, s2, M), (s2, s2, M) \), and \( (s3, s2, M) \), for any multiset of messages \( M \)
Applicable receive events

- Any receive event \( e=(c,m,d) \in R_{pi} \) is said to be applicable in a configuration \( C=(c_{p1}, ..., c_{pi}, ..., c_{pn}, M) \) if \( c_{pi}=c \) and \( m \in M \).

- If event \( e \) is applied, we get \( e(C)=(c_{p1}, ..., d, ..., c_{pn}, M-\{m\}) \).

- Example, if there are 2 processors, \( p1, p2 \), and
  - Set of states on \( p1 \) and \( p2 \) are \( Z_1=Z_2=\{s1, s2, s3\} \)
  - \( R_{p1}=\{ (s1, m, s3), (s3, n, s2), (s2, o, s3) \} \)
  - \( R_{p2}=\{ (s1, n, s2), (s2, m, s3), (s3, o, s1) \} \)
  - The receive event \( (s2, m, s3) \in R_{p2} \) on \( p2 \) is applicable in all the following configurations:
    - \( (s1, s2, M), (s2, s2, M), \) and \( (s3, s2, M) \), for every multiset of messages \( M \) which contains the message \( m \) destined to \( p2 \)
Synchronous message passing

- … we now know how to model asynchronous message passing

- Synchronous message passing is similar, but easier. Read the book!
Fairness?

- Notice; several events at different processes might be applicable in a given state
  - Which one should be applied first?

  - If \( e_1, e_2, e_3 \) are applicable at a given time, should we assume they should be applied one after another \( e_1, e_2, e_3 \)?

  - Maybe such an assumption would be too strong? We would like our model to not assume too much!

  - We definitely do not want a have a stupid system where event \( e_2 \) is always applicable, but never applied!

- In other words, we sometimes want to assume some fairness in the possible executions of our model
Weakly Fair and Strongly Fair executions

- An execution is *weakly fair*, if it is guaranteed that it can never happen that an event it applicable infinitely many times after each other (consecutively), without ever occurring in the execution.

- An execution is *strongly fair*, if it is guaranteed that it can never happen that an event it applicable infinitely many times (not necessarily consecutively), without ever occurring in the execution.
Proving correctness of an algorithm

- Given our formal model, we can now proof that an algorithm is correct.

- Most interesting properties in distributed systems fall into one of these two categories:
  - Safety requirements
  - Liveness properties
Safety and Liveness

- A safety requirement requires that some property holds in every execution in \textit{each} reachable configuration.

- A liveness requirement requires that some property will hold in every execution for \textit{some} configuration which is reachable.
We will make an assertion of a property which we are interested to proof

An assertion is a predicate on configurations of the distributed system

- I.e. a function which takes a configuration as input and returns either true or false
- $P(\delta)$ is true or false
Safety Properties as assertions

- We proof a safety property by showing that its assertion is always true.

- For an STS $S = (C, \rightarrow, I)$ we write $\{P\} \rightarrow \{Q\}$ to denote that for each transition $\gamma \rightarrow \delta$ in $S$ we have that:
  - if $P(\gamma) = \text{true}$, then $Q(\delta) = \text{true}$
An invariant property

- We say that a property $P$ is **invariant** in $S=(C,\rightarrow, I)$ if
  1. For all initial configurations $\gamma \in I$, $P(\gamma) = true$
  2. $\{P\} \rightarrow \{P\}$ for all configurations

**Theorem**
If $P$ is invariant in $S$, then $P$ is **true** in each configuration in every execution

**Proof:**
For any execution in $S$, $(\gamma_1, \gamma_2, \gamma_3...)$, then we know by the definition of an execution that $\gamma_1$ is an initial configuration, and by condition 1 of an invariant $P(\gamma_1)$ is true
By induction, condition 2 of an invariant gives that $P(\gamma_k)$ is true for every $k \geq 2$
Liveness properties as Assertions

- Let *term* be an assertion which is only true if a configuration is terminal, and false otherwise.

- To show a liveness property $P$, we want $P$ to be true for some configuration in every execution.

- We say that a system $S$ terminates properly if *term* is true, then also $P$ is true.

- A partial order $\langle W, < \rangle$ is well-founded if there is no infinite $w_1 > w_2 > w_3 \ldots$, where $w_i \in W$.
  - Example: all positive natural numbers.
Norm functions

A norm function from the set of configurations $C$ to a well-founded set $W$ is a function $f$ such that if $\gamma \rightarrow \delta$, for $\delta, \gamma \in C$, then $f(\gamma) > f(\delta)$ or $P(\delta) = true$.

Example of a norm function for property $P$:
- Configurations $C = \{X_0, X_1, X_2, X_3\}$
- Configuration transitions $\rightarrow = \{X_2 \rightarrow X_1, X_2 \rightarrow X_0, X_0 \rightarrow X_3\}$
- Norm function $f = \{X_2 \rightarrow 100, X_1 \rightarrow 50, X_2 \rightarrow 100, X_0 \rightarrow 1\}$
- It is a norm function for this system because
  - $X_2 \rightarrow X_1$, and $f(X_2) = 100 > f(X_1) = 50$
  - $X_2 \rightarrow X_0$, and $f(X_2) = 100 > f(X_0) = 1$
  - $X_0 \rightarrow X_3$, and $P(X_3) = true$
Theorem:

In a transition system $S$, which properly terminates for $P$, and a norm function $f$ for $P$ exists, then $P$ is a liveness property which will be true for some configuration in every execution.
Summary

- Distributed systems can be formally modeled by state transition systems STS
  - Asynchronous message passing
  - Synchronous message passing

- The model can be used to:
  - Prove impossibility results, i.e. statements about all possible algorithms in the system
  - Prove that a certain property always holds (safety)
  - Prove that a certain property will hold in every execution for some configurations