An introduction to Concurrency Theory
Communicating and Mobile Systems

Frank D. Valencia

French National Center for Scientific Research (CNRS)
Computer Science Laboratory of École Polytechnique de Paris (LIX)

Oct 2005/CLEI’05
Concurrent Systems

Multiple agents (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

Example

E.g., *The Internet* (a complex one!)
Concurrency is everywhere...

**Concurrent Systems**

Multiple **agents** (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

**Example**

E.g., *The Internet* (a complex one!)
Concurrency is everywhere...

Concurrent Systems

Multiple agents (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

Example

E.g., *The Internet* (a complex one!)
Concurrency is everywhere...

**Concurrent Systems**

Multiple *agents* (*processes*) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

**Example**

E.g., *The Internet* (a complex one!)
Concurrency is everywhere...

**Concurrent Systems**

Multiple *agents* (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

**Example**

E.g., *The Internet* (a complex one!)
Concurrency is everywhere...

Concurrent Systems

Multiple agents (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

Example

E.g., The Internet (a complex one!)
Concurrent Systems

Multiple agents (processes) that interact among each other.

Some fundamental concurrent systems:

- Reactive Systems.
- Synchronous Communicating Systems
- Mobile Systems.

Concurrent systems can combine:

Example

E.g., *The Internet* (a complex one!)
Concurrency: A Serious Challenge...

Models of Concurrency

Formal Models to describe and analyze concurrent systems.
Concurrency: A Serious Challenge...

Models of Concurrency

Formal Models to describe and analyze concurrent systems.

As other models of reality, these models must be

- be simple
- be expressive,
- be formal
- provide reasoning techniques.

Just like $\lambda$—calculus model of sequential computation!
Concurrent computation is usually

- Non-Terminating
- Reactive (or Interactive)
- Nondeterministic (Unpredictable).
Concurrency: A Serious Challenge...

Models of Concurrency

Formal Models to describe and analyze concurrent systems.

In concurrency theory

- Each model focuses in a fundamental phenomenon: E.g., Synchrony and Mobility.
- But there is no yet a “cannonical model”.

...Probably because concurrency is a very broad (young) area.
Concurrency: A Serious Challenge...

Models of Concurrency

Formal Models to describe and analyze concurrent systems.

Some well established model of concurrency:

- Process Calculi (Like λ-calculi)
  - CCS (Synchronous communication)
  - π-calculus (CCS Extension to Mobility)
- Petri Nets (Like Automata)
Mobility

What kind of *mobility* will we consider in this tutorial?

- Processes move.
- Links move✓

The last one, because it is the $\pi$-calculus’ choice; for flexibility and simplicity.

- $\pi$-calculus has the ability of sending private and public *links* (*names*).
Mobility

What kind of *mobility* will we consider in this tutorial?

- Processes move.
- **Links** move ✓

The last one, because it is the π-calculus’ choice; for flexibility and simplicity.

- π-calculus has the ability of sending private and public *links* (*names*).
Mobility

What kind of mobility will we consider in this tutorial?

- Processes move.
- **Links** move √

The last one, because it is the π-calculus’ choice; for flexibility and simplicity.

- π-calculus has the ability of sending private and public links (**names**).
Link Mobility

Mobile phones
Link Mobility

Mobile phones
A flavour of the pi-calculus

In the $\pi$ calculus:

$$(\nu a)(\overline{ba}.S \parallel a(x).P) \parallel b(c).\overline{cd}.C$$
In the pi-calculus:

$$(\nu a)(\overline{ba}.S \parallel a(x).P) \parallel b(c).\overline{c}d.C \rightarrow S \parallel (\nu a)(a(x).P \parallel \overline{a}d.C)$$
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Classic Automata Theory

Automata

Definition

An automata $A$ over an alphabet $Act$ is a tuple $(Q, Q_0, Q_f, T)$ where

- $S(A) = Q = \{q_0, q_1, \ldots\}$ is the set of states,
- $S_0(A) = Q_0 \subseteq Q$ is the set of initial states,
- $S_f(A) = Q_f \subseteq Q$ is the set of accepting (or final) states,
- $T(A) = T \subseteq Q \times Act \times Q$ is the set of transitions.

Write $q \xrightarrow{a} q'$ for $(q, a, q') \in T$. Omit $Act$ when it is understood.
Example: A Typical Graph Representation
Definition (Acceptance, Regularity)

- A accepts $a_1 ... a_n$ iff there is a path $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ in $T(A)$ with $q_0 \in S_o(A)$ and $q_n \in S_f(A)$.
- The Language of (or recognized by) $A$, $L(A)$, is the set of sequences accepted by $A$.
- Regular sets are those recognized by finite-state automata (FSA).
Definition (Acceptance, Regularity)

- **A accepts** $a_1...a_n$ iff there is a path $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ in $T(A)$ with $q_0 \in S_o(A)$ and $q_n \in S_f(A)$.
- The **Language** of (or recognized by) $A$, $L(A)$, is the set of sequences accepted by $A$.
- **Regular sets** are those recognized by *finite-state automata* (FSA).

Fact

Regular Expressions (e.g. $a.(b + c)^*$) are as expressive as FSA.
An string accepted: 10001.
The language accepted: All binary strings with an even number of 1’s.
A Corresponding Regular Expression: $0^* (10^*10^*)^*$
Fact

1. Deterministic and NonDeterministic FSA are equally expressive.
2. Regular sets are closed under (a) union, (b) complement, (c) intersection.

Exercises

1. Prove 2.b and 2.c.
2. Prove that emptiness problem of a given FSA is decidable.
3. Prove that language equivalence of two given FSA is decidable.
Classic Automata Theory
Some Nice Properties and Exercises

Fact

1. **Deterministic and NonDeterministic FSA are equally expressive.**
2. Regular sets are **closed** under (a) union, (b) complement, (c) intersection.

Exercises

1. Prove 2.b and 2.c.
2. Prove that emptiness problem of a given FSA is **decidable**.
3. Prove that language equivalence of two given FSA is **decidable**.
Classic Automata Theory is solid and foundational with several applications in computer science. BUT...
Classic Automata Theory is solid and foundational with several applications in computer science. BUT...

Example: A FSA Vending Machine Representation
Classic Automata Theory

What is good for?

Classic Automata Theory is solid and foundational with several applications in computer science. BUT...

Example: A FSA Vending Machine Representation

\[ L(B_1) = L(B_2) \] but \( B_2 \) would represent an annoying vending machine!
Classic Automata Theory
What is good for?

Classic Automata Theory is solid and foundational with several applications in computer science. BUT...

Fact

Language equivalence may be too weak for interactive behavior.
Classic Automata Theory.

The Problem with Language Equivalence

In automata theory $a \cdot (b + c) = a \cdot b + a \cdot c$. I.e., $p_0$ and $q_0$ are equivalent.

We need stronger equivalences that does not validate the above.
Outline

1. **From Computability to Concurrency Theory**
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. **Calculus of Communicating Systems CCS**
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. **Mobility and the pi calculus**
   - Syntax and Semantics
   - Applications
   - Equivalences
Bisimilarity: The Equivalence of Reactive Systems.

Transition Systems

Transition systems are just automata in which initial and final states are irrelevant.
Bisimilarity: The Equivalence of Reactive Systems.

**Simulation**

Let $T$ be transition system. A relation $R \subseteq S(T) \times S(T)$ is a *simulation* iff for each $(p, q) \in R$:

- If $p \xrightarrow{a} p'$ then there is $q'$ s.t. $q \xrightarrow{a} q'$ and $(p', q') \in R$. 

CLEI’05 Cali  Communicating and Mobile Systems
Bisimilarity: The Equivalence of Reactive Systems.

Bisimulation

A relation $R$ is a **bisimulation** iff $R$ and its converse $R^{-1}$ are both simulations.
Bisimilarity: The Equivalence of Reactive Systems.

Similaridad & Bisimilaridad

▷ $p$ simulates $q$ iff there exists a simulation $R$ s.t. $(p, q) \in R$.
▷ $p$ and $q$ are bisimilar, written $p \sim q$, if there exists a bisimulation $R$ s.t. $(p, q) \in R$. 
Bisimilarity: The Equivalence of Reactive Systems.

Example: Trace Equivalence is not Bisimilarity

Here $p_0$ simulates $q_0$ but $q_0$ does not simulate $p_0$, so $p_0$ and $q_0$ are not bisimilar.
Example: Bisimilarity is not graph equivalence

Bisimilarity equates LTS with different graph structure: E.g., $p_0$ and $q_0$ are bisimilar. But how do we prove they are bisimilar?
Bisimilarity: The Equivalence of Reactive Systems.

Example: Proving Bisimilarity

To prove that $p_0$ and $q_0$ are bisimilar simply exhibit a bisimulation $R$. E.g. $R = \{(p_0, q_0), (p_0, q_2), (p_1, q_1), (p_2, q_1)\}$. 
Example: Proving Bisimilarity

To prove that $p_0$ and $q_0$ are bisimilar simply exhibit a bisimulation $R$. E.g. $R = \{(p_0, q_0), (p_0, q_2), (p_1, q_1), (p_2, q_1)\}$. 
Example: Bisimilarity and simulation

Suppose that $p$ simulates $q$ and $q$ simulates $p$. Are $p$ and $q$ necessarily bisimilar?
Bisimilarity: The Equivalence of Reactive Systems.

Example: Bisimilarity and simulation

No! Here \( p \) simulates \( q \) and \( q \) simulates \( p \) but they are not bisimilar. Notice that \( p \) is less \textit{reliable} than \( q \); i.e., it \textit{deadlocks} after \( a \).
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Process calculi treat processes much like $\lambda$-calculus treats computable functions:

- Structure of terms represent the structure of processes. E.g.:

\[ P \parallel Q \]

- Operational Semantics to represent computational steps. E.g.:

\[
\text{PAR} \quad \frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q}
\]
Process calculi treat processes much like λ-calculus treats computable functions:

- Structure of terms represent the structure of processes. E.g.:
  \[ P \parallel Q \]

- Operational Semantics to represent computational steps. E.g.:
  \[
  \frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q}
  \]

\(P\parallel Q\)
Process Calculi

Process calculi treat processes much like \( \lambda \)-calculus treats computable functions:

- Structure of terms represent the structure of processes. E.g.:

  \[ P \parallel Q \]

- Operational Semantics to represent computational steps. E.g.:

  \[
  \begin{align*}
  \text{PAR} & \quad \frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q}
  \end{align*}
  \]
Typical Issues in Developing Process Calculi

- **Which process constructs fit the intended phenomena?**
  E.g. atomic actions, parallelism, nondeterminism.

- **How should these constructs be given meaning?**
  E.g. Operational Semantics, Denotational Semantics.

- **How should processes be compared?**
  E.g. Observable Behavior, Process Equivalences.

- **How should process properties be specified and proved?**
  E.g. Logic for expressing process specifications (like in Hoare’s Logic).

- **How expressive is the calculus?**
  E.g. Expressiveness of a nice fragment of the calculus.
Typical Issues in Developing Process Calculi

- **Which process constructs fit the intended phenomena?**
  E.g. atomic actions, parallelism, nondeterminism.

- **How should these constructs be given meaning?**
  E.g. Operational Semantics, Denotational Semantics.

- **How should processes be compared?**
  E.g. Observable Behavior, Process Equivalences.

- **How should process properties be specified and proved?**
  E.g. Logic for expressing process specifications (like in Hoare’s Logic).

- **How expressive is the calculus?**
  E.g. Expressiveness of a nice fragment of the calculus.
Typical Issues in Developing Process Calculi

- **Which process constructs fit the intended phenomena?**
  E.g. atomic actions, parallelism, nondeterminism.

- **How should these constructs be given meaning?**
  E.g. Operational Semantics, Denotational Semantics.

- **How should processes be compared?**
  E.g. Observable Behavior, Process Equivalences.

- **How should process properties be specified and proved?**
  E.g. Logic for expressing process specifications (like in Hoare’s Logic).

- **How expressive is the calculus?**
  E.g. Expressiveness of a nice fragment of the calculus.
Typical Issues in Developing Process Calculi

- **Which process constructs fit the intended phenomena?**
  E.g. atomic actions, parallelism, nondeterminism.

- **How should these constructs be given meaning?**
  E.g. Operational Semantics, Denotational Semantics.

- **How should processes be compared?**
  E.g. Observable Behavior, Process Equivalences.

- **How should process properties be specified and proved?**
  E.g. Logic for expressing process specifications (like in Hoare’s Logic).

- **How expressive is the calculus?**
  E.g. Expressiveness of a nice fragment of the calculus.
Typical Issues in Developing Process Calculi

- **Which process constructs fit the intended phenomena?**
  E.g. atomic actions, parallelism, nondeterminism.

- **How should these constructs be given meaning?**
  E.g. Operational Semantics, Denotational Semantics.

- **How should processes be compared?**
  E.g. Observable Behavior, Process Equivalences.

- **How should process properties be specified and proved?**
  E.g. Logic for expressing process specifications (like in Hoare’s Logic).

- **How expressive is the calculus?**
  E.g. Expressiveness of a nice fragment of the calculus.
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
     - Bisimilarity
     - Observable Behaviour and other Equivalences
     - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Basic Entities of CCS:

- A set $\mathcal{N} = a, b, \ldots$ of **names** and $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ of **co-names**.
- A set $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$ of **labels** (ranged over by $l, l', \ldots$).
- A set $\text{Act} = \mathcal{L} \cup \{\tau\}$ of **actions** (ranged over by $a, b, \ldots$).
- Action $\tau$ is called the **silent or unobservable action**.
- Actions $a$ and $\overline{a}$ are viewed as being “complementary”. i.e., $a = \overline{\overline{a}}$. 
Definition (CCS Syntax of Processes)

\[ P, Q, \ldots := 0 \mid a.P \mid P \parallel Q \mid P + Q \mid (\nu a)P \mid A(a_1, \ldots, a_n) \]

- **Bound names** of \( P \), \( fn(P) \): Those with a bound occurrence in \( P \).
- **Free names of** \( P \), \( bn(P) \): Those with a not bound occurrence in \( P \).
- For each (call) \( A(a_1, \ldots, a_n) \) there is a unique **process definition** \( A(b_1 \ldots b_n) = P \), with \( fv(P) \subseteq \{ b_1, \ldots, b_n \} \).
- The set of all processes is denoted by \( \mathcal{P} \).
# CCS: Operational Semantics

<table>
<thead>
<tr>
<th>ACT</th>
<th>$a.P \xrightarrow{a} P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM₁</td>
<td>$P \xrightarrow{a} P'$</td>
</tr>
<tr>
<td>SUM₂</td>
<td>$Q \xrightarrow{a} Q'$</td>
</tr>
<tr>
<td>COM₁</td>
<td>$P \xrightarrow{a} P'$</td>
</tr>
<tr>
<td>COM₂</td>
<td>$Q \xrightarrow{a} Q'$</td>
</tr>
<tr>
<td>COM₃</td>
<td>$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\tau} Q'}{P</td>
</tr>
</tbody>
</table>
| RES         | $\frac{P \xrightarrow{a} P'}{(
u a)P \xrightarrow{a} (\nu a)P'}$ if $a \neq a$ and $a \neq \overline{a}$ |
| REC         | $\frac{P_A[b_1 \ldots b_n/a_1 \ldots a_n] \xrightarrow{a} P'}{A \langle b_1 \ldots b_n \rangle \xrightarrow{a} P'}$ if $A(a_1 \ldots a_n) \overset{\text{def}}{=} P_A$ |
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Labeled Transition System of CCS

The LTS of CCS has \( P \) as its states and its transitions given by the operational semantics. Define \( P \sim Q \) iff the states corresponding to \( P \) and \( Q \) are bisimilar.
CCS: Bisimilarity

Labeled Transition System of CCS

The LTS of CCS has $P$ as its states and its transitions given by the operational semantics. Define $P \sim Q$ iff the states corresponding to $P$ and $Q$ are bisimilar.

Example: Processes and LTS states

$P = a.(b.0 + c.0)$ corresponds to $p_0$ and $Q = a.b.0 + a.c.0$ corresponds to $q_0$. Hence $P \not\sim Q$. 
Labeled Transition System of CCS

The LTS of CCS has $P$ as its states and its transitions given by the operational semantics. Define $P \sim Q$ iff the states corresponding to $P$ and $Q$ are bisimilar.

Some Basic Equalities

- $P \parallel Q \sim Q \parallel P$
- $P \parallel 0 \sim P$
- $(P \parallel Q) \parallel R \sim P \parallel (Q \parallel R)$
- $(\nu a)0 \sim 0$
- $P \parallel (\nu a)Q \sim (\nu a)(P \parallel Q)$
- $(\nu a)P \sim (\nu b)P[b/a]$
CCS: The expansion law.

- Notice that $a.0 \parallel b.0$ is bisimilar to $a.b.0 + b.a.0$.
- More generally, we have the expansıon law:

$$(\nu \bar{a})(P_1 \parallel \ldots \parallel P_n) \sim$$

$${\Sigma}\{a_i.(\nu \bar{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_n \mid P_i \xrightarrow{a_i} P_i', \bar{a}, a_i \not\in \bar{a}\} + \Sigma\{\tau.(\nu \bar{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_j' \ldots \parallel P_n \mid P_i \xrightarrow{\backslash} P_i', P_j \xrightarrow{\Dagger} P_j'\}$$

- So, every move in $(\nu \bar{a})(P_1 \parallel \ldots \parallel P_n)$ is either one of the $P_i$ or a communication between some $P_i$ and $P_j$. 
CCS: The expansion law.

- Notice that $a.0 \parallel b.0$ is bisimilar to $a.b.0 + b.a.0$.
- More generally, we have the **expansion law**:
  \[(ν\bar{a})(P_1 \parallel \ldots \parallel P_n) \sim \]
  \[
  \Sigma\{a_i.(ν\bar{a})(P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_n \mid P_i \xrightarrow{a_i} P_i', \bar{a}, a_i \not\in \bar{a})\} + \\
  \Sigma\{τ.(ν\bar{a})(P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_j \ldots \parallel P_n \mid P_i \xrightarrow{l} P_i', P_j \xrightarrow{\bar{l}} P_j'\}\]

- So, every move in $(ν\bar{a})(P_1 \parallel \ldots \parallel P_n)$ is either one of the $P_i$ or a communication between some $P_i$ and $P_j$. 
CCS: The expansion law.

- Notice that \( a.0 \parallel b.0 \) is bisimilar to \( a.b.0 + b.a.0 \).

- More generally, we have the **expansion law**:

\[
(\nu \overline{a})(P_1 \parallel \ldots \parallel P_n) \sim \\
\sum\{a_i.(\nu \overline{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_n \mid P_i \xrightarrow{a_i} P_i', \overline{a}, a_i \not\in \overline{a})
\]

\[
\quad + \sum\{\tau.(\nu \overline{a})(P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_j' \ldots \parallel P_n \mid P_i \xrightarrow{l} P_i', P_j \xrightarrow{\overline{l}} P_j')\}
\]

- So, every move in \((\nu \overline{a})(P_1 \parallel \ldots \parallel P_n)\) is either one of the \(P_i\) or a communication between some \(P_i\) and \(P_j\).
Congruence issues

- Suppose that \( P \sim Q \). We would like \( P \parallel R \sim Q \parallel R \).
- More generally we would like

\[
C[P] \sim C[Q]
\]

where \( C[.] \) is a process context. I.e., we want \( \sim \) to be a congruence.
- The notion of congruence allows us to replace “equals with equals”.
- In fact, \( \sim \) is a congruence. How can we prove this?
Congruence issues

- Suppose that $P \sim Q$. We would like $P \parallel R \sim Q \parallel R$.
- More generally we would like

$$C[P] \sim C[Q]$$

where $C[.]$ is a process context. I.e., we want $\sim$ to be a congruence.

- The notion of congruence allows us to replace “equals with equals”.
- In fact, $\sim$ is a congruence. How can we prove this?
Suppose that $P \sim Q$. We would like $P \parallel R \sim Q \parallel R$.

More generally we would like

\[ C[P] \sim C[Q] \]

where $C[.]$ is a process context. I.e., we want $\sim$ to be a congruence.

The notion of congruence allows us to replace “equals with equals”.

In fact, $\sim$ is a congruence. How can we prove this?
Congruence issues

- Suppose that $P \sim Q$. We would like $P \parallel R \sim Q \parallel R$.
- More generally we would like

$$C[P] \sim C[Q]$$

where $C[.]$ is a process context. I.e., we want $\sim$ to be a congruence.
- The notion of congruence allows us to replace “equals with equals”.
- In fact, $\sim$ is a congruence. How can we prove this?
In principle, $P$ and $Q$ should be equivalent iff another process (the environment, an observer) cannot observe any difference in their behavior.

Notice $\tau.P \not\sim P$, although $\tau$ is an unobservable action. So $\sim$ could be too strong.

We look for other notion of equivalence focused in terms of observable actions (i.e., actions $\xrightarrow{l}$, $l \in L$).
Observable Behavior

- In principle, $P$ and $Q$ should be equivalent iff another process (the environment, an observer) cannot observe any difference in their behavior.

- Notice $\tau.P \not\sim P$, although $\tau$ is an unobservable action. So $\sim$ could be too strong.

- We look for other notion of equivalence focused in terms of observable actions (i.e., actions $\xrightarrow{l}$, $l \in L$).
Observable Behavior

- In principle, $P$ and $Q$ should be equivalent iff another process (the environment, an observer) cannot observe any difference in their behavior.

- Notice $\tau.P \not\sim P$, although $\tau$ is an unobservable action. So $\sim$ could be too strong.

- We look for other notion of equivalence focused in terms of observable actions (i.e., actions $\xrightarrow{l}$, $l \in \mathcal{L}$).
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Observations

- Think of any $\xrightarrow{l}$ with $l \neq \tau$ as an observation.
- Think of an experiment $e$ as a sequence $l_1.l_2\ldots.l_n$ of observable actions.
- Notation: If $s = a_1 \ldots a_n \in \text{Act}^*$ then define
  \[ s \xrightarrow{e} = (\xrightarrow{\tau})^* a_1 (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* a_n (\xrightarrow{\tau})^* \]
- Notice that $\xrightarrow{e}$ for $e = l_1.l_2\ldots.l_n \in \mathcal{L}$ denotes a sequence of observable actions inter-spread with $\tau$ actions: *The notion of experiment.*
Think of any $l \rightarrow$ with $l \neq \tau$ as an **observation**.

Think of an **experiment** $e$ as a sequence $l_1 \cdot l_2 \ldots l_n$ of observable actions.

**Notation:** If $s = a_1 \ldots a_n \in Act^*$ then define

$$s \Rightarrow = (\tau \rightarrow)^* a_1 (\tau \rightarrow)^* \ldots (\tau \rightarrow)^* a_n (\tau \rightarrow)^*$$

Notice that $e \Rightarrow$ for $e = l_1 . l_2 \ldots l_n \in L$ denotes a sequence of observable actions inter-spread with $\tau$ actions: **The notion of experiment**.
Observations

- Think of any $l \rightarrow$ with $l \neq \tau$ as an observation.
- Think of an experiment $e$ as a sequence $l_1.l_2 \ldots l_n$ of observable actions.

**Notation:** If $s = a_1 \ldots a_n \in Act^*$ then define

$$\xrightarrow{s} = (\tau)^* \xrightarrow{a_1} (\tau)^* \ldots (\tau)^* \xrightarrow{a_n} (\tau)^*$$

- Notice that $\xrightarrow{e}$ for $e = l_1.l_2 \ldots l_n \in \mathcal{L}$ denotes a sequence of observable actions inter-spread with $\tau$ actions: The notion of experiment.
Observations

- Think of any $\frac{l}{l \neq \tau}$ with $l \neq \tau$ as an observation.
- Think of an experiment $e$ as a sequence $l_1.l_2 \ldots l_n$ of observable actions.

**Notation:** If $s = a_1 \ldots a_n \in Act^*$ then define

\[
\xrightarrow{s} = (\frac{\tau}{\tau})^* \xrightarrow{a_1} (\frac{\tau}{\tau})^* \ldots (\frac{\tau}{\tau})^* \xrightarrow{a_n} (\frac{\tau}{\tau})^*
\]

- Notice that $\xrightarrow{e}$ for $e = l_1.l_2 \ldots l_n \in L$ denotes a sequence of observable actions inter-spread with $\tau$ actions: **The notion of experiment.**
Trace Equivalence

Definition (Trace Equivalence)

$P$ and $Q$ are *trace equivalent*, written $P \sim_t Q$, iff for every experiment (here called trace) $e = l_1 \ldots l_n \in L^*$

$$P \xrightarrow{e} \iff Q \xrightarrow{e}$$

Observations:

- $\tau.P \sim_t P$ (nice!)
- $a.b.0 + a.c.0 \sim_t a.(b.0 + c.0)$ (not that nice:)
- $a.b.0 + a.0 \sim_t a.b.0$ (not sensitive to deadlocks)
- $a.0 + b.0 \sim_t (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)$
Trace Equivalence

Definition (Trace Equivalence)

\[ P \text{ and } Q \text{ are } \textit{trace equivalent}, \text{ written } P \sim_t Q, \text{ iff for every experiment (here called trace) } e = l_1 \ldots l_n \in \mathcal{L}^* \]

\[ P \xrightarrow{e} \iff Q \xrightarrow{e} \]

Observations:

- \( \tau.P \sim_t P \) (nice!)
- \( a.b.0 + a.c.0 \sim_t a.(b.0 + c.0) \) (not that nice:)
- \( a.b.0 + a.0 \sim_t a.b.0 \) (not sensitive to deadlocks)
- \( a.0 + b.0 \sim_t (\nu c)(c.0 \parallel c.a.0 \parallel \overline{c}.b.0) \)
Failures Equivalence

Definition (Failures Equivalence)

A pair \((e, L)\), where \(e \in \mathcal{L}^*\) (a \textit{trace}) and \(L \subseteq \mathcal{L}\), is a \textit{failure} for \(P\) iff there is \(P'\) s.t.:

\[
(1) P \xrightarrow{e} P', \quad (2) P' \nrightarrow l \text{ for all } l \in L, \quad \text{and} \quad (3) P' \nrightarrow^f_	au
\]

\(P \sim_f Q\) iff \(P\) and \(Q\) have the same failures.

Observations:

- \(\tau.P \sim_f P\).
- \(a.b.0 + a.c.0 \nrightarrow_f a.(b.0 + c.0)\) (Exercise: why?)
- \(a.b.0 + a.0 \nrightarrow_f a.b.0\).
- \(a.0 + b.0 \nrightarrow_f (\nu c)(c.0 \parallel c.a.0 \parallel c.b.0)\) (Exercise)
- \(a.(b.c.0 + b.d.0) \sim_f a.b.c.0 + a.b.d.0\).
- \(\exists \text{ a process} \equiv \cdot.\). We have \(\equiv_0 \nrightarrow_f \equiv_0\).
Failures Equivalence

Definition (Failures Equivalence)

A pair \((e, L)\), where \(e \in \mathcal{L}^*\) (a trace) and \(L \subset \mathcal{L}\), is a failure for \(P\) iff there is \(P'\) s.t.:

\[
(1) P \xrightarrow{e} P', \quad (2) P' \not\xrightarrow{l} \text{ for all } l \in L, \quad \text{and } (3) P' \not\xrightarrow{\tau}
\]

\(P \sim_f Q\) iff \(P\) and \(Q\) have the same failures.

Observations:

- \(\tau.P \sim_f P\).
- \(a.b.0 + a.c.0 \not\sim_f a.(b.0 + c.0)\) (Exercise: why?)
- \(a.b.0 + a.0 \not\sim_f a.b.0\).
- \(a.0 + b.0 \not\sim_f (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)\) (Exercise)
- \(a.(b.c.0 + b.d.0) \sim_f a.b.c.0 + a.b.d.0\).
- Let \(P \equiv \sigma_D\). We have \(\tau.0 \not\sim_f \sigma_D\).
Weak Bisimilarity (Observational Equivalence)

Definition (Weak Bisimilarity)

A *symmetric* relation $R$ is a *weak bisimulation* iff for every $(P, Q) \in R$:

- If $P \xrightarrow{e} P'$ then there is $Q'$ s.t. $Q \xrightarrow{e} Q'$ and $(P', Q') \in R$.

Define $P \approx Q$ iff $(P, Q) \in R$ for some *weak bisimulation* $R$.

Observations:

- $\tau.P \approx P$, $a.\tau.P \approx a.P$
- However, $a.0 + b.0 \not\approx a.0 + \tau.b.0$
- $a.(b.c.0 + b.d.0) \not\approx a.b.c.0 + a.b.d.0$ (Exercise)
- Let $D = \tau.D$. We have $\tau.0 \approx D$.
- $a.0 + b.0 \not\approx (\nu c)(c.0 \parallel \overline{c}.a.0 \parallel \overline{c}.b.0)$ (Exercise)
**Weak Bisimilarity (Observational Equivalence)**

**Definition (Weak Bisimilarity)**

A *symmetric* relation $R$ is a *weak bisimulation* iff for every $(P, Q) \in R$:

> If $P \xrightarrow{e} P'$ then there is $Q'$ s.t. $Q \xrightarrow{e} Q'$ and $(P', Q') \in R$.

Define $P \approx Q$ iff $(P, Q) \in R$ for some *weak bisimulation* $R$.

**Observations:**

- $\tau.P \approx P$, $a.\tau.P \approx a.P$
- However, $a.0 + b.0 \not\approx a.0 + \tau.b.0$
- $a.(b.c.0 + b.d.0) \not\approx a.b.c.0 + a.b.d.0$ (Exercise)
- Let $D = \tau.D$. We have $\tau.0 \approx D$.
- $a.0 + b.0 \not\approx (\nu c)(c.0 || \overline{c}.a.0 || \overline{c}.b.0)$ (Exercise)
Process Logic: Verification and Specification

- Processes can be used to *specify and verify* behavioural properties:
  E.g., $2p.\overline{tea}.0 + 2p.\overline{coffe}.0$ specifies a machine which does not satisfy the behaviour specified by a $2p.(\overline{tea}.0 + \overline{coffe}.0)$.

- Alternatively, a logic can be used. An example of a property is “$P$ will never not execute a bad action” or “$P$ eventually executes execute a good action”.
Process Logic: Verification and Specification

Processes can be used to specify and verify behavioural properties:
E.g., \(2p.\text{tea}.0 + 2p.\text{coffe}.0\) specifies a machine which does not satisfy the behaviour specified by \(2p.(\text{tea}.0 + \text{coffe}.0)\).

Alternatively, a logic can be used. An example of a property is “\(P\) will never not execute a bad action” or “\(P\) eventually executes execute a good action”.
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Hennessy & Milner Logic

**Definition (Logic Syntax)**

\[ F \ := \ \text{true} \mid \text{false} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \langle K \rangle F \mid [K]F \]

where \( K \) is a set of actions

- The boolean operators are interpreted as in propositional logic.
- \( \langle K \rangle F \) (possibility) asserts (of a given \( P \)): It is possible for \( P \) to do an action \( a \in K \) and then evolve into a state \( Q \) that satisfies \( F \).
- \( [K]F \) (necessity) asserts (of a given \( P \)): If \( P \) can do an action \( a \in K \) then it must evolve into a state \( Q \) which satisfies \( F \).
Hennessy & Milner Logic

Definition (Logic Syntax)

\[ F := \text{true} \mid \text{false} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \langle K \rangle F \mid [K]F \]

where \( K \) is a set of actions

- The boolean operators are interpreted as in propositional logic.
- \( \langle K \rangle F \) (possibility) asserts (of a given \( P \)): It is possible for \( P \) to do a \( a \in K \) and then evolve into a \( Q \) that satisfy \( F \).
- \( [K]F \) (necessity) asserts (of a given \( P \)): If \( P \) can do a \( a \in K \) then it must evolve into a \( Q \) which satisfies \( F \).
Hennessy&Milner Logic

Definition (Logic Syntax)

\[ F \equiv \text{true} \mid \text{false} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \langle K \rangle F \mid [K]F \]

where \( K \) is a set of actions

- The boolean operators are interpreted as in propositional logic.
- \( \langle K \rangle F \) (possibility) asserts (of a given \( P \)): It is possible for \( P \) to do a \( a \in K \) and then evolve into a \( Q \) that satisfy \( F \).
- \([K]F\) (necessity) asserts (of a given \( P \)): If \( P \) can do a \( a \in K \) then it must evolve into a \( Q \) which satisfies \( F \).
Hennessy & Milner Logic

**Definition (Logic Syntax)**

\[ F := \text{true} \mid \text{false} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \langle K \rangle F \mid [K]F \]

where \( K \) is a set of actions

- The boolean operators are interpreted as in propositional logic.
- \( \langle K \rangle F \) (possibility) asserts (of a given \( P \)): It is possible for \( P \) to do a \( a \in K \) and then evolve into a \( Q \) that satisfy \( F \).
- \( [K]F \) (necessity) asserts (of a given \( P \)): If \( P \) can do a \( a \in K \) then it must evolve into a \( Q \) which satisfies \( F \).
Definition

The compliance of $P$ with the specification $F$, written $P \models F$, is given by:

- $P \not\models \text{false}$
- $P \models \text{true}$
- $P \models F_1 \land F_2$ iff $P \models F_1$ and $P \models F_2$
- $P \models F_1 \lor F_2$ iff $P \models F_1$ or $P \models F_2$
- $P \models \langle K \rangle F$ iff for some $Q$, $P \xrightarrow{a} Q$, $a \in K$ and $Q \models F$
- $P \models [K]F$ iff if $P \xrightarrow{a} Q$ and $a \in K$ then $Q \models F$
From Computability to Concurrency Theory
Calculus of Communicating Systems CCS
Mobility and the pi calculus
Summary

Hennessy & Milner Logic and Bisimilarity

Example

Let $P_1 = a.(b.0 + c.0)$, $P_2 = a.\textbf{b}.0 + a.\textbf{c}.0$. Also let

$$F = \langle\{a\}\rangle(\langle\{b\}\rangle \text{true} \land \langle\{c\}\rangle \text{true}).$$

Notice that $P_1 \models F$ but $P_2 \not\models F$.

Theorem

$P \sim Q$ if and only, for every $F$, $P \models F$ iff $Q \models F$.
Hennessy & Milner Logic and Bisimilarity

Example

Let $P_1 = a.(b.0 + c.0)$, $P_2 = a.b.0 + a.c.0$. Also let

$$F = \langle\{a\}\rangle (\langle\{b\}\rangle \text{true} \land \langle\{c\}\rangle \text{true}).$$

Notice that $P_1 \models F$ but $P_2 \not\models F$.

Theorem

$P \sim Q$ if and only, for every $F$, $P \models F$ iff $Q \models F$. 
A Linear Temporal Logic.

Definition (Logic Syntax)

\[ F := \text{true} \mid \text{false} \mid L \mid F_1 \lor F_2 \mid F_1 \land F_2 \mid \Diamond F \mid \Box F \]

where \( L \) is a set of non-silent actions.

- Formulae assert properties of traces.
- Boolean operators are interpreted as usual.
- \( L \) asserts (of a given trace \( s \)) that the first action of \( s \) must be in \( L \cup \{\tau\} \).
- \( \Diamond F \) asserts (of a given trace \( s \)) that at some point in \( s \), \( F \) holds.
- \( \Box F \) asserts (of a given trace \( s \)) that at every point in \( s \), \( F \) holds.
A Linear Temporal Logic.

**Definition (Logic Syntax)**

\[ F ::= \text{true} \mid \text{false} \mid L \mid F_1 \lor F_2 \mid F_1 \land F_2 \mid \Diamond F \mid \Box F \]

where \( L \) is a set of non-silent actions.

- Formulae assert properties of traces.
- Boolean operators are interpreted as usual
- \( L \) asserts (of a given trace \( s \)) that the first action of \( s \) must be in \( L \cup \{\tau\} \)
- \( \Diamond F \) asserts (of a given trace \( s \)) that at some point in \( s \), \( F \) holds.
- \( \Box F \) asserts (of a given trace \( s \)) that at every point in \( s \), \( F \) holds.
A Linear Temporal Logic.

Definition (Logic Syntax)

\[ F := \text{true} \mid \text{false} \mid L \mid F_1 \lor F_2 \mid F_1 \land F_2 \mid \Diamond F \mid \Box F \]

where \( L \) is a set of non-silent actions.

- Formulae assert properties of traces.
- Boolean operators are interpreted as usual.
- \( L \) asserts (of a given trace \( s \)) that the first action of \( s \) must be in \( L \cup \{\tau\} \).
- \( \Diamond F \) asserts (of a given trace \( s \)) that at some point in \( s \), \( F \) holds.
- \( \Box F \) asserts (of a given trace \( s \)) that at every point in \( s \), \( F \) holds.
A Linear Temporal Logic.

Definition (Logic Syntax)

\[ F := \text{true} \mid \text{false} \mid L \mid F_1 \lor F_2 \mid F_1 \land F_2 \mid \Diamond F \mid \Box F \]

where \( L \) is a set of non-silent actions.

- Formulae assert properties of traces.
- Boolean operators are interpreted as usual.
- \( L \) asserts (of a given trace \( s \)) that the first action of \( s \) must be in \( L \cup \{\tau\} \).
- \( \Diamond F \) asserts (of a given trace \( s \)) that at some point in \( s \), \( F \) holds.
- \( \Box F \) asserts (of a given trace \( s \)) that at every point in \( s \), \( F \) holds.
A Linear Temporal Logic.

**Definition (Logic Syntax)**

\[ F ::= \text{true} \mid \text{false} \mid L \mid F_1 \lor F_2 \mid F_1 \land F_2 \mid \Diamond F \mid \Box F \]

where \( L \) is a set of non-silent actions.

- Formulae assert properties of traces.
- Boolean operators are interpreted as usual.
- \( L \) asserts (of a given trace \( s \)) that the first action of \( s \) must be in \( L \cup \{\tau\} \).
- \( \Diamond F \) asserts (of a given trace \( s \)) that at some point in \( s \), \( F \) holds.
- \( \Box F \) asserts (of a given trace \( s \)) that at every point in \( s \), \( F \) holds.
Semantics of the temporal logic.

**Definition**

An infinite sequence of actions $s = a_1.a_2 \ldots$ satisfies (or is a model of) $F$, written $s \models F$, iff $\langle s, 1 \rangle \models F$, where

\[
\begin{align*}
\langle s, i \rangle &\models \text{true} \\
\langle s, i \rangle &\not\models \text{false} \\
\langle s, i \rangle &\models L \quad \text{iff} \quad a_i \in L \cup \tau \\
\langle s, i \rangle &\models F_1 \lor F_2 \quad \text{iff} \quad \langle s, i \rangle \models F_1 \text{ or } \langle s, i \rangle \models F_2 \\
\langle s, i \rangle &\models F_1 \land F_2 \quad \text{iff} \quad \langle s, i \rangle \models F_1 \text{ and } \langle s, i \rangle \models F_2 \\
\langle s, i \rangle &\models \Box F \quad \text{iff} \quad \text{for all } j \geq i \quad \langle s, j \rangle \models F \\
\langle s, i \rangle &\models \Diamond F \quad \text{iff} \quad \text{there is a } j \geq i \text{ s.t. } \langle s, j \rangle \models F
\end{align*}
\]

$P \models F$ iff whenever $P \xrightarrow{s} \text{ then } \hat{s} \models F$, where $\hat{s} = s.\tau.\tau.\ldots$. 
Temporal Logic and Trace Equivalence

Example

Consider the trace equivalent processes

\[ A(a, b, c) \overset{\text{def}}{=} a.(b.A(a, b, c) + c.A(a, b, c)) \] and

\[ B(a, b, c) \overset{\text{def}}{=} a.b.B(a, b, c) + a.c.B(a, b, c), \] and the property

\[ F = \Box \Diamond (b \lor c). \]

We have \( A(a, b, c) \models F \) and \( B(a, b, c) \models F \).

Theorem

\( P \sim_t Q \) then for every linear temporal formula \( F \),

\[ P \models F \iff Q \models F \]

Question. Does the other direction of the theorem hold?
Temporal Logic and Trace Equivalence

Example

Consider the trace equivalent processes

\[ A(a, b, c) \stackrel{\text{def}}{=} a.(b.A(a, b, c) + c.A(a, b, c)) \] and

\[ B(a, b, c) \stackrel{\text{def}}{=} a.b.B(a, b, c) + a.c.B(a, b, c) \]

and the property

\[ F = □◊(b ∨ c) \]

We have \( A(a, b, c) \models F \) and \( B(a, b, c) \models F \).

Theorem

\[ P \sim_t Q \text{ then for every linear temporal formula } F, \]

\[ P \models F \iff Q \models F \]

Question. Does the other direction of the theorem hold?
Example

Consider the trace equivalent processes
\[ A(a, b, c) \overset{\text{def}}{=} a.(b.A(a, b, c) + c.A(a, b, c)) \] and
\[ B(a, b, c) \overset{\text{def}}{=} a.b.B(a, b, c) + a.c.B(a, b, c), \]
and the property
\[ F = \Box \Diamond (b \lor c). \]
We have \( A(a, b, c) \models F \) and \( B(a, b, c) \models F \).

Theorem

\[ P \sim_t Q \text{ then for every linear temporal formula } F, \]
\[ P \models F \iff Q \models F \]

Question. Does the other direction of the theorem hold?
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Mobility and the pi calculus

What kind of *mobility* are we talking about?

- Processes move.
- Links move ✓

The last one, because it is the $\pi$-calculus’ choice; for flexibility and simplicity.

- $\pi$-calculus has the ability of sending private and public links (*names*).
Mobility and the pi calculus

What kind of *mobility* are we talking about?

- Processes move.
- **Links** move

The last one, because it is the \( \pi \)-calculus’ choice; for flexibility and simplicity.

- \( \pi \)-calculus has the ability of sending private and public *links* (*names*).
What kind of *mobility* are we talking about?

- Processes move.
- *Links* move

The last one, because it is the $\pi$-calculus’ choice; for flexibility and simplicity.

- $\pi$-calculus has the ability of sending private and public *links* (*names*).
Link Mobility

Mobile phones
Link Mobility

Mobile phones

![Diagram of mobile phones connected to a control system](image-url)
Definition (Syntax)

\[ P, Q, \ldots := P \parallel Q \mid \sum_{i \in I} \alpha . P \mid (\nu a) P \mid !P \mid \text{if } a = b \text{ then } P \]

where \( \alpha := \tau \mid \bar{ab} \mid a(x) \).

- Names=Channels=Ports=Links.
- \( \bar{ab}.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \( (\nu a)P \) : “create a fresh name \( a \) private to \( P \)”
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : “replicate \( P \)” i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
\[ P, Q, \ldots := P \parallel Q \mid \sum_{i \in I} \alpha.P \mid (\nu a)P \mid !P \mid \text{if } a = b \text{ then } P \]

where \( \alpha := \tau \mid \bar{a}b \mid a(x) \).

- **Names=Channels=Ports=Links.**
- \( \bar{a}b.P \) : "send \( b \) on channel \( a \) and then activate \( P \)"
- \( a(x).P \) : "receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)"
- \( (\nu a)P \) : "create a fresh name \( a \) private to \( P \)"
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : "replicate \( P \)" i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
\( \pi \)-Calculus: Syntax

**Definition (Syntax)**

\[
P, Q, \ldots := P \parallel Q \mid \Sigma_{i \in I} \alpha.P \mid (\nu a)P \mid !P \mid \text{if } a = b \text{ then } P
\]

where \( \alpha := \tau \mid \bar{a}b \mid a(x). \)

- **Names=Channels=Ports=Links.**
- \( \bar{a}b.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \( (\nu a)P \) : “create a fresh name \( a \) private to \( P \)”
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : “replicate \( P \)” i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
\[ P, Q, \ldots := P \parallel Q \mid \sum_{i \in I} \alpha.P \mid (\nu a)P \mid \!P \mid \text{if } a = b \text{ then } P \]

where \( \alpha := \tau \mid \bar{a}b \mid a(x) \).

- Names=Channels=Ports=Links.
- \( \bar{a}b.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \( (\nu a)P \) : “create a fresh name \( a \) private to \( P \)”
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : “replicate \( P \)” i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
\[ P, Q, \ldots : = P \parallel Q \mid \Sigma_{i \in I} \alpha.P \mid (\nu a)P \mid !P \mid \text{if } a = b \text{ then } P \]

where \( \alpha : = \tau \mid \bar{ab} \mid a(x) \).

- Names=Channels=Ports=Links.
- \( \bar{ab}.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \( (\nu a)P \) : “create a fresh name a private to \( P \)”
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : “replicate \( P \)” i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
\[ P, Q, \ldots := P \parallel Q \mid \Sigma_{i \in I} \alpha.P \mid (\nu a)P \mid !P \mid \text{if } a = b \text{ then } P \]

where \( \alpha := \tau \mid \bar{a}b \mid a(x) \).

- Names=Channels=Ports=Links.
- \( \bar{a}b.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \((\nu a)P\) : “create a fresh name \( a \) private to \( P \)”
- \((\nu a)P\) and \( a(x).Q \) are the only binders.
- \(!P\) : “replicate \( P \)” i.e., \(!P\) represents \( P \parallel P \parallel P \parallel \ldots \)
π—Calculus: Syntax

Definition (Syntax)

\[ P, Q, \ldots := P \parallel Q \mid \Sigma_{i \in I} \alpha. P \mid (\nu a)P \mid !P \mid \text{if } a = b \text{ then } P \]

where \( \alpha := \tau \mid \bar{a}b \mid a(x). \)

- Names=Channels=Ports=Links.
- \( \bar{a}b.P \) : “send \( b \) on channel \( a \) and then activate \( P \)”
- \( a(x).P \) : “receive a name on channel \( a \) (if any), and replace \( x \) with it in \( P \)”
- \( (\nu a)P \) : “create a fresh name \( a \) private to \( P \)”
- \( (\nu a)P \) and \( a(x).Q \) are the only binders.
- \( !P \) : “replicate \( P \)” i.e., \( !P \) represents \( P \parallel P \parallel P \parallel \ldots \)
A flavour of the pi-calculus

In the $\pi$ calculus:

$$(\nu a) (\bar{b}a. S \parallel a(x). P) \parallel b(c). \bar{c}d. C$$
In the pi-calculus:

\[(\nu a)(\bar{b}a.S \parallel a(x).P) \parallel b(c).\bar{c}d.C \rightarrow S \parallel (\nu a)(a(x).P \parallel \bar{a}d.C)\]
Mobile Exercises

Write an agent that:

- reads something from port $a$ and sends it twice along port $b$
- generates \textit{infinitely many} different names—and send them along channel $a$.
- encodes biadic-comunication: $\bar{x}z_1z_2.P$ which outputs both $y_1$ and $y_2$ on channel $x$, and $x(y_1,y_2).Q$ which inputs two names sent on channel $x$. 
Mobile Exercises

Write an agent that:

- reads something from port $a$ and sends it twice along port $b$

- generates *infinitely many* different names—and send them along channel $a$.

- encodes biadic communication: $\bar{x}z_1z_2.P$ which outputs both $y_1$ and $y_2$ on channel $x$, and $x(y_1,y_2).Q$ which inputs two names sent on channel $x$. 
Mobile Exercises

Write an agent that:

- reads something from port $a$ and sends it twice along port $b$
- generates *infinitely many* different names—and send them along channel $a$.
- encodes biadic-communication: $\bar{x}z_1z_2.P$ which outputs both $y_1$ and $y_2$ on channel $x$, and $x(y_1, y_2).Q$ which inputs two names sent on channel $x$. 
Reaction Semantics of $\pi$.

The reactive semantics of $\pi$ consists of an *structural congruence* $\equiv$ and the *reactive rules*.

- The relation $\equiv$ describes irrelevant syntactic aspects.
- The reactive semantics describes evolutions due to synchronous communication.
Reaction Semantics of $\pi$.

The reactive semantics of $\pi$ consists of an *structural congruence* $\equiv$ and the *reactive rules*.

- The relation $\equiv$ describes *irrelevant syntactic aspects*
- The reactive semantics describes evolutions due to *synchronous* communication.
Reaction Semantics of $\pi$.

The reactive semantics of $\pi$ consists of an *structural congruence* $\equiv$ and the *reactive rules*.

- The relation $\equiv$ describes irrelevant syntactic aspects.
- The reactive semantics describes evolutions due to *synchronous* communication.
Structural Equivalence

Structural Congruence

*Structural Congruence* $\equiv$ is the congruence given by:

- $P \equiv Q$ if $P$ can be $\alpha$-converted into $Q$.
- $P \parallel 0 \equiv P$, $P \parallel Q \equiv Q \parallel P$, $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$.
- $(\nu a)0 = 0, (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$,
- $(\nu a)(P \parallel Q) \equiv P \parallel (\nu a)Q$ if $a \notin fn(P)$,
- $!P \equiv P \parallel !P$.

Example

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$?
2. $x(y).y(z) \equiv x(z).y(y)$?
3. $x \parallel y \equiv x.y + y.x$?
Structural Equivalence

Structural Congruence $\equiv$ is the congruence given by:

- $P \equiv Q$ if $P$ can be $\alpha$-converted into $Q$.
- $P \parallel 0 \equiv P$, $P \parallel Q \equiv Q \parallel P$, $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$.
- $(\nu a)0 = 0, (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$,
- $(\nu a)(P \parallel Q) \equiv P \parallel (\nu a)Q$ if $a \notin fn(P)$.
- $!P \equiv P \parallel !P$.

Example

1. $(\nu a)P \equiv P$ if $a \notin fn(P)$?
2. $x(y).y(z) \equiv x(z).y(y)$?
3. $x \parallel y \equiv x.y + y.x$?
Reactive Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU</td>
<td>$\tau.P + M \rightarrow P$</td>
</tr>
<tr>
<td>REACT</td>
<td>$(a(x).P + M) \parallel (\bar{a}.Q + N) \rightarrow P[b/x] \parallel Q$</td>
</tr>
<tr>
<td>PAR</td>
<td>$P \rightarrow P'$</td>
</tr>
<tr>
<td></td>
<td>$P \parallel Q \rightarrow P' \parallel P$</td>
</tr>
<tr>
<td>RES</td>
<td>$P \rightarrow P'$</td>
</tr>
<tr>
<td></td>
<td>$(\nu a)P \rightarrow (\nu a)P'$</td>
</tr>
<tr>
<td>STRUCT</td>
<td>$P \equiv P' \rightarrow Q' \equiv Q$</td>
</tr>
<tr>
<td></td>
<td>$P \rightarrow Q$</td>
</tr>
</tbody>
</table>

Exercise. Assume $a \not\in fn(S) \cup fn(C)$. Write a derivation for our Server example:

$$(\nu a)(\bar{b}.a.S \parallel a(x).P) \parallel b(c).\bar{c}d.C \rightarrow S \parallel (\nu a)(a(x).P \parallel \bar{a}d.C)$$
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Applications: Persistent and Mutable Data Structures

Definition

A cell is basis for defining mutable and persistent data structures.

\[
\text{Cell}(u, r, z) \overset{\text{def}}{=} u(z').\text{Cell}(u, r, z') + r(x).\overline{x}z.\text{Cell}(u, r, z)
\]

The cell current contents \(z\) can be updated via \(u\) or read via \(r\).

But we do not have recursive definitions in the \(\pi\)-calculus. Can the above cell be rewritten in with the \(\pi\)-calculus primitives?
Applications: Persistent and Mutable Data Structures

Definition

A cell is a basis for defining mutable and persistent data structures.

\[ \text{Cell}(u, r, z) \overset{\text{def}}{=} u(z').\text{Cell}(u, r, z') + r(x).\overline{x}z.\text{Cell}(u, r, z) \]

The cell current contents \( z \) can be updated via \( u \) or read via \( r \).

But we do not have recursive definitions in the \( \pi \)-calculus. Can the above cell be rewritten in with the \( \pi \)-calculus primitives?
Recursive Definitions

We wish invocations $A(z)$ to recursive definition (as in CCS) $A(x) \overset{\text{def}}{=} P$. Do we need to extend the calculus?

No! Just take an $a \not\in \text{fn}(P)$:

1. Replace in $P$ every invocation $A(u)$ with $\bar{a}u$. Call $\hat{P}$ the resulting processes.

2. Encode $A(x) = P$ as $!a(x) . \hat{P}$.

3. Encode $A(z)$ as $(\nu a)(\bar{a}(z) \parallel !a(x) . \hat{P})$. 
Recursive Definitions

We wish invocations $A(z)$ to recursive definition (as in CCS) $A(x) \overset{\text{def}}{=} P$. Do we need to extend the calculus?

No! Just take an $a \not\in \text{fn}(P)$:

1. Replace in $P$ every invocation $A(u)$ with $\bar{a}u$. Call $\hat{P}$ the resulting processes.

2. Encode $A(x) = P$ as $!a(x).\hat{P}$.

3. Encode $A(z)$ as $(\nu a)(\bar{a}(z) \parallel !a(x).\hat{P})$. 
Recursive Definitions

We wish invocations $A(z)$ to recursive definition (as in CCS) $A(x) \overset{\text{def}}{=} P$. Do we need to extend the calculus?

No! Just take an $a \notin fn(P)$:

1. Replace in $P$ every invocation $A(u)$ with $\bar{a}u$. Call $\hat{P}$ the resulting processes.

2. Encode $A(x) = P$ as $!a(x).\hat{P}$.

3. Encode $A(z)$ as $(\nu a)(\bar{a}(z) || !a(x).\hat{P})$. 
Recursive Definitions

We wish invocations \( A(z) \) to recursive definition (as in CCS) \( A(x) \overset{\text{def}}{=} P \). Do we need to extend the calculus?

No! Just take an \( a \notin fn(P) \):

1. Replace in \( P \) every invocation \( A(u) \) with \( \bar{a}u \). Call \( \hat{P} \) the resulting processes.

2. Encode \( A(x) = P \) as \( !a(x).\hat{P} \).

3. Encode \( A(z) \) as \((\nu a)(\bar{a}(z) \parallel !a(x).\hat{P})\).
Should $P = x(z) \parallel \bar{y}u$ be equivalent to $Q = x(z).\bar{y}u + \bar{y}u.x(z)$ in π—like in CCS?

- Take $R = a(x).P$ and $S = a(x).Q$.
- Hence, $\bar{a}y$ can tell $R$ and $S$ apart! I.e., $\bar{a}y \parallel R$.

Passing links around complicates matters—not unduly, though:). Need to consider substitutions, contexts, etc!
π Equivalences

Should $P = x(z) \parallel \overline{y}u$ be equivalent to $Q = x(z).\overline{y}u + \overline{y}u.x(z)$ in $\pi$—like in CCS?

- Take $R = a(x).P$ and $S = a(x).Q$.
- Hence, $\overline{a}y$ can tell $R$ and $S$ apart! I.e., $\overline{a}y \parallel R$.

Passing links around complicates matters—not unduly, though:). Need to consider substitutions, contexts, etc!
Should $P = x(z) || \bar{y}u$ be equivalent to $Q = x(z).\bar{y}u + \bar{y}u.x(z)$ in $\pi$—like in CCS?

- Take $R = a(x).P$ and $S = a(x).Q$.

- Hence, $\bar{a}y$ can tell $R$ and $S$ apart! I.e., $\bar{a}y || R$.

Passing links around complicates matters—not unduly, though:). Need to consider substitutions, contexts, etc!
Outline

1. From Computability to Concurrency Theory
   - Basic Concepts from Automata Theory
   - Bisimilarity Equivalence

2. Calculus of Communicating Systems CCS
   - General Aspects of Process Calculi
   - Syntax and Semantics
   - Bisimilarity
   - Observable Behaviour and other Equivalences
   - Verification and Specification

3. Mobility and the pi calculus
   - Syntax and Semantics
   - Applications
   - Equivalences
Barbed Equivalence

Intuitively we write $P \downarrow_x (P \downarrow_{\overline{x}})$ iff $P$ can receive (send) a link on channel $x$.

Formally, $P \downarrow_l$ where $l \in \mathcal{L}$ is given by:

- $x(y).P \downarrow_x$.
- $\overline{x}z.P \downarrow_{\overline{x}}$.
- $\Sigma_{i \in I} \alpha_i.P; \downarrow_l$ iff for some $i \in I$, $\alpha_i.P; \downarrow_l$
- $P \parallel Q \downarrow_l$ iff either $P \downarrow_l$ or $Q \downarrow_l$
- $(\nu z)P \downarrow_l$ iff $l \neq z$, $\overline{l} \neq z$ and $P \downarrow_l$
Barbed Equivalence

Definition

$R$ is a **barbed simulation** iff for every $(P, Q) \in R$:
1. If $P \xrightarrow{} P'$ then $\exists Q'$: $Q \xrightarrow{} Q' \land (P', Q') \in R$.
2. If $P \downarrow$ then $Q \downarrow$.

Definition

**Barbed bisimilarity**: $P \sim_B Q$ iff there is $R$ such that $R$ and $R^{-1}$ are barbed simulations and $(P, Q) \in R$.

Example

Let $P = x(z) \parallel \bar{y}u$ and $Q = x(z).\bar{y}u + \bar{y}u.x(z)$. Then $P \sim_B Q$ and $a(x).P \sim_B a(x).Q$. 
Barbed Equivalence

Definition

*R is a barbed simulation* iff for every \((P, Q) \in R\):

1. If \(P \xrightarrow{} P'\) then \(\exists Q': Q \xrightarrow{} Q' \land (P', Q') \in R\).
2. If \(P \downarrow_1\) then \(Q \downarrow_1\).

Example

Let \(P = x(z) \parallel \bar{y}u\) and \(Q = x(z).\bar{y}u + \bar{y}u.x(z)\). Then \(P \sim_B Q\) and \(a(x).P \sim_B a(x).Q\).
Barbed Equivalence

**Definition**

*R is a *barbed simulation* iff for every \((P, Q) \in R:\)*

1. If \(P \rightarrow P'\) then \(\exists Q': Q \rightarrow Q' \land (P', Q') \in R\).
2. If \(P \downarrow\) then \(Q \downarrow\).

**Definition**

*Barbed bisimilarity:* \(P \sim_B Q\) iff there is \(R\) such that \(R\) and \(R^{-1}\) are barbed simulations and \((P, Q) \in R\).

**Example**

Let \(P = x(z) \parallel y u\) and \(Q = x(z).y u + y u.x(z)\). Then \(P \sim_B Q\) and \(a(x).P \sim_B a(x).Q\).
Let $\approx_B$ be the congruence *induced* by $\sim_B$. I.e:

**Definition**

$$P \approx_B Q \iff \text{for every context } C[\cdot], C[P] \sim_B C[Q]$$

**Example**

Note that $x(z) \parallel \check{y}u \not\approx_B x(z).\check{y}u + \check{y}u.x(z)$. 
Let \( \approx_B \) be the congruence induced by \( \sim_B \). I.e:

**Definition**

\[
P \approx_B Q \text{ iff } \text{ for every context } C[.], \ C[P] \sim_B C[Q]
\]

**Example**

Note that \( x(z) \parallel \bar{y}u \not\approx_B x(z).\bar{y}u + \bar{y}u.x(z) \).
A Labelled Semantics for Pi.

Transitions $P \xrightarrow{\alpha} Q$ where $\alpha \in \{\overline{ab}, a(x), \tau, \overline{a} \nu u\}$ are given by:

**PREFIX**

$\alpha . P + M \xrightarrow{\alpha} P$

**PAR**

$\frac{P \xrightarrow{\alpha} P', bn(\alpha) \cap fn(Q) = \emptyset}{P | Q \xrightarrow{\alpha} P' | Q}$

**COM**

$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}u} Q'}{P | Q \xrightarrow{\tau} P' \{u/x\} | Q'}$

**RES**

$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$

**OPEN**

$\frac{P \xrightarrow{\overline{a}x} P', a \neq x}{(\nu x)P \xrightarrow{\overline{a}uv} P'}$

**CLOSE**

$\frac{P \xrightarrow{a(u)} P', Q \xrightarrow{\overline{a}vu} Q'}{P | Q \xrightarrow{\tau} (\nu u)(P' | Q')}$
Early Bisimilarity for Pi

Definition

$R$ is a **simulation** iff for $\forall (P, Q) \in R$:
If $P \xrightarrow{\alpha} P'$ then
(1) If $\alpha$ is an input $a(x)$ then $\forall u, \exists Q'$:
$Q \xrightarrow{\alpha} Q' \land (P'[u/x], Q'[u/x]) \in R$
(2) If $\alpha$ is not an input then $\exists Q : Q \xrightarrow{\alpha} Q' \land (P', Q') \in R$

Definition

**(Early) Bisimilarity**: $P \sim_E Q$ iff there is $R$ such $R$ and $R^{-1}$ are simulations and $(P, Q) \in R$.

Example: Let $P = x(z) \parallel \bar{y}u$ and $Q = x(z).\bar{y}u + \bar{y}u.x(z)$. Then $a(x).P \not\sim_E a(x).Q$ but $P \sim_E Q$. 
Early Congruence for Pi.

**Definition**

Define $P \approx_E Q$ iff $P\sigma \sim_E Q\sigma$ for all substitutions $\sigma$.

**Example:** Note that $x(z) \parallel \bar{y}u \not\approx_E x(z).\bar{y}u + \bar{y}u.x(z)$. In fact, the relation $\approx_E$ is a congruence. Furthermore $\approx_E = \approx_B$.
Thanks

Example
On first slide.

Example
On second slide.
Thanks

Example
On first slide.

Example
On second slide.
Theorem

On first slide.

Corollary

On second slide.
Theorem

On first slide.

Corollary

On second slide.
Theorem

In left column.

Corollary

In right column.
New line
Make Titles Informative.

**Theorem**

*In left column.*

**Corollary**

*In right column.*

*New line*
Summary

- The **first main message** of your talk in one or two lines.
- The **second main message** of your talk in one or two lines.
- Perhaps a **third message**, but not more than that.

**Outlook**

- What we have not done yet.
- Even more stuff.