A Process Calculus

- Description of process networks
  - Static communication topologies.
- History sketch
  - Robin Milner, 1980.
  - CCS: Calculus of Communicating Systems.
  - Various revisions and elaborations.
  - Extended to mobile processes ($\pi$-calculus).
- Algebraic approach
  - Concurrent system modeled by term.
  - Theory of term manipulations.
  - External behavior preserved.
- Observational equivalence
  - External communications follow same pattern.
  - Internal behavior may differ.

Modeling of communication and concurrency.
A Simple Example

- **Agent $C$**
  - Dynamic system is network of agents.
  - Each agent has own identity persisting over time.
  - Agent performs actions (external communications or internal actions).
  - Behavior of a system is its (observable) capability of communication.

- **Agent has labeled ports.**
  - Input port $\text{in}$.
  - Output port $\overline{\text{out}}$.

- **Behavior of $C$:**
  - $C := \text{in}(x).C'(x)$
  - $C'(x) := \overline{\text{out}}(x).C$
Behavior Descriptions

- Agent names can take parameters.
- Prefix `in(x)`
  - Handshake in which value is received at port `in` and becomes the value of variable `x`.
- Agent expression `in(x).C'(x)`
  - Perform handshake and proceed according to definition of `C'`.
- Agent expression `out(x).C`
  - Output the value of `x` at port `out` and proceed according to the definition of `C`.
- Scope of local variables:
  - `Input` prefix introduces variable whose scope is the agent expression `C`.
  - Formal parameter of defining equation introduces variable whose scope is the equation.
Behavior Descriptions

\[ C := \text{in}(x) . \overline{\text{out}}(x) . C \]
\[ A := \text{in}(x) . \text{in}(y) . \overline{\text{out}}(x) . \overline{\text{out}}(y) . A \]

- How do behaviors differ?
  - \( A \) inputs two values and outputs two values.
  - \( C \) inputs and output a single value.

\[
\begin{array}{c}
\text{in} \\
C \\
\text{out}
\end{array}
\]

- Agent expression \( C \bowtie C \).
  - \textit{Combinator} \( A_1 \bowtie A_2 \) (defined later).
  - Agent formed by linking \( \overline{\text{out}} \) of \( A_2 \) to \( \text{in} \) of \( A_1 \).
  - \( \bowtie \) is associative.
Bounded Buffer

- $C^{(n)}$
  - $C^{(n)} := C \odot C \odot \ldots \odot C$
  - Behaves as bounded buffer of capacity $n$.
  - $C^{(n)} = \text{Buff}_n$

- Specification $\text{Buff}_n(s)$
  - $\text{Buff}_n \langle \rangle := \text{in}(x).\text{Buff}_n \langle x \rangle$
  - $\text{Buff}_n \langle v_1, \ldots, v_n \rangle := \text{out}(v_n).\text{Buff}_n \langle v_1, \ldots, v_{n-1} \rangle$
  - $\text{Buff}_n \langle v_1, \ldots, v_k \rangle := \text{in}(x).\text{Buff}_n \langle x, v_1, \ldots, v_k \rangle + \text{out}(v_k).\text{Buff}_n \langle v_1, \ldots, v_{k-1} \rangle(0 < k < n)$

- $C^{(n)} = \text{Buff}_n \langle \rangle$
Summation

- Basic combinator ’+’
  - $P + Q$ behaves like $P$ or like $Q$.
  - When one performs its first action, other is discarded.
  - If both alternatives are allowed, selection is non-deterministic.

- Combining forms
  - *Summation* $P + Q$ of two agents.
  - *Sequencing* $\alpha. P$ of action $\alpha$ and agent $P$.

- Different levels of abstractions
  - Agent can be expressed directly in terms of its interaction with environment $(C, \textit{Buff}_n)$.
  - Agent can be expressed indirectly in terms of its composition of sammer agents $(C^{(n)})$. 

Example

- Received values to be acknowledged.
  - $D := \text{in}(x).\overline{\text{out}(x)}.\text{ackout.}\overline{\text{ackin}.D}$
  - $D$ acknowledges input after it has delivered value as output and received acknowledgement.
  - Synchronization actions $\text{ackout, } \overline{\text{ackin}}.$

- Combination of $n$ copies of $D$:
  - $D^{(n)} := D \bowtie D \bowtie \ldots \bowtie D$
  - $D^{(n)}$ behaves like single copy of $D$!
  - $D \bowtie D = D!$

- Alternative definition:
  - $D' := \text{in}(x).\overline{\text{ackin}}.\overline{\text{out}(x)}.\text{ackout}.D'$
  - $D'^{(n)} = \text{Buff'}_n \langle \rangle$.
  - Slightly modified specification $\text{Buff'}_n$.
Examples

- A vending machine:
  - Big chocolate costs 2p, small one costs 1p.
  - $V := 2p.\text{big} . \text{collect} . V + 1p.\text{little} . \text{collect} . V$

- A multiplier
  - $Twice := \text{in}(x) . \text{out}(2 \times x) . Twice.$
  - Output actions may take expressions.
A Process Calculus I

A Larger Example: The Jobshop

- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - *Jobs* arrive sequentially on a belt.
  - A job is to drive a peg into a block.
Flow Graphs

- Ports may be linked to more than one other port.
  - Jobbers compete for use of hammer.
  - Jobbers compete for use of job.
  - Source of non-determinism.

- Ports of belt are omitted from system.
  - in and out are external.

- Internal ports are not labelled:
  - Ports by which jobbers acquire and release tools.

- Flow graph is exact mathematical object:
  - Homomorphism from flow graph algebra to behavior algebra.
The Tools

- Behaviors:
  - \( \text{Hammer} := \text{geth. Busyhammer} \)
    \( \text{Busyhammer} := \text{puth. Hammer} \)
  - \( \text{Mallet} := \text{geth. Busymallet} \)
    \( \text{Busymallet} := \text{puth. Mallet} \)

- \textbf{Sort} = \text{set of labels}
  - \( P : L \ldots \text{agent } P \text{ has sort } L \)
  - \( \text{Hammer}: \{\text{geth, puth}\} \)
    \( \text{Mallet}: \{\text{getm, putm}\} \)
    \( \text{Jobshop}: \{\text{in, out}\} \)
The Jobbers

- Different kinds of jobs:
  - Easy jobs done with hands.
  - Hard jobs done with hammer.
  - Other jobs done with hammer or mallet.
- Behavior:
  - $Jobber := \text{in}(job).\text{Start}(job)$
  - $\text{Start}(job) := \textbf{if} \ \text{easy}(job) \ \textbf{then} \ \text{Finish}(job)$
    $\textbf{else if} \ \text{hard}(job) \ \textbf{then} \ \text{Uhammer}(job)$
    $\textbf{else} \ \text{Usetool}(job)$
  - $\text{Usetool}(job) := \text{Uhammer}(job) + \text{Umallet}(job)$
  - $\text{Uhammer}(job) := \overline{\text{geth}}.\text{puth}.\text{Finish}(job)$
  - $\text{Umallet}(job) := \overline{\text{getm}}.\text{putm}.\text{Finish}(job)$
  - $\text{Finish}(job) := \overline{\text{out}}(\text{done}(job)).\text{Jobber}$
Composition of Agents

- **Jobber-Hammer** subsystem
  - *Jobber* | *Hammer*
  - *Composition* operator |
  - Agents may proceed independently or interact through *complementary* ports.
  - Join complementary ports.

- **Two jobbers sharing hammer:**
  - *Jobber* | *Hammer* | *Jobber*
  - Composition is commutative and associative.
Further Composition

- **Internalisation** of ports:
  - No further agents may be connected to ports:
  - *Restriction* operator \(\backslash\)
  - \(\backslash L\) internalizes all ports \(L\).
  - \(\text{Jobber} \mid \text{Jobber} \mid \text{Hammer})\backslash\{\text{getm,putm}\}

- **Complete system**:
  - \(\text{Jobshop} := (\text{Jobber} \mid \text{Jobber} \mid \text{Hammer} \mid \text{Malllet})\backslash L\)
  - \(L := \{\text{getm,putm,putm,putm}\}\)
Reformulations

• Alternative formulation:
  - \( ((\text{Jobber} \mid \text{Jobber} \mid \text{Hammer}) \backslash \{\text{geth, puth}\} \mid \text{Mallet}) \backslash \{\text{getm, putm}\} \)
  - \textit{Algebra} of combinators with certain laws of equivalence.

• \textit{Relabelling} Operator
  - \( P[l_1'/l_1, \ldots, l_n'/l_n] \)
  - \( f(l) = f(l) \)

• Semaphore agent
  - \( \text{Sem} := \text{get}.\text{put}.\text{Sem} \)

• Reformulation of tools
  - \( \text{Hammer} := \text{Sem}[\text{geth}/\text{get}, \text{puth}/\text{put}] \)
  - \( \text{Mallet} := \text{Sem}[\text{getm}/\text{get}, \text{putm}/\text{put}] \)
Equality of Agents

- Five basic operators:
  - Prefix: $\alpha.P$
  - Summation: $P + Q$
  - Composition: $P | Q$
  - Restriction: $P \backslash \{l_1, \ldots, l_n\}$
  - Relabelling: $P[l'_1/l_1, \ldots, l'_n/l_n]$

- **Strongjobber** only needs hands:
  - $\text{Strongjobber} := \text{in}(\text{job}).\overline{\text{out}(\text{done}(\text{job}))}.\text{Strongjobber}$

- Claim:
  - $\text{Jobshop} = \text{Strongjobber} | \text{Strongjobber}$
  - Specification of system Jobshop
  - Proof of equality required.
Action and Transition

• Names and co-names
  – Set $A$ of names ($\text{geth}, \text{ackin}, \ldots$)
  – Set $\overline{A}$ of co-names ($\overline{\text{geth}}, \overline{\text{ackin}}, \ldots$)
  – Set of labels $L = A \cup \overline{A}$

• Transition $P \xrightarrow{l} Q$
  – $\text{Hammer} \xrightarrow{\text{geth}} \text{Busyhammer}$
  – $\text{Busyhammer} \xrightarrow{\text{puth}} \text{Hammer}$

• Agents $A$ and $B$

$$
\begin{array}{c}
\bullet A \quad \bullet \overline{c} \\
\bullet B \quad \bullet b
\end{array}
$$

  – $A := a.A', A' := \overline{c}.A$
  – $B := c.B', B' := \overline{b}.B$
Composite Agents

- Composite Agent \( A|B \)

\[
\begin{array}{c}
\bullet A \quad \bar{c} \quad B \quad \bar{b}
\end{array}
\]

- \( A \xrightarrow{a} A' \) allows \( A|B \xrightarrow{a} A'|B \)
- \( A' \xrightarrow{\bar{c}} A \) allows \( A'|B \xrightarrow{\bar{c}} A|B \)
- \( A' \xrightarrow{\bar{c}} A \) and \( B \xrightarrow{c} B' \) allows \( A'|B \xrightarrow{\tau} A|B' \)

- **Completed (perfect) action** \( \tau \).
  - Simultaneous action of both agents.
  - *Internal* to composed agent.
  - \( \text{Act} = L \cup \{\tau\} \)

- **Internal versus external actions**
  - Internal actions are ignored.
  - Only external actions are visible.
  - Two systems are *equivalent* if they exhibit same pattern of external actions.
  - \( P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n \) equivalent to \( P \xrightarrow{\tau} P_n \)
Restrictions

- Restriction $(A|B)\setminus c$

- $P \xrightarrow{\alpha} P'$ allows $P\setminus L \xrightarrow{\alpha} P'\setminus L$
  (if $\alpha$, $\overline{\alpha}$ not in $L$)

- Transition (derivation) tree

\[
\begin{array}{c}
(A|B)\setminus c \\
\downarrow a \\
(A'|B)\setminus c \\
\downarrow \tau \\
(A|B')\setminus c \\
\leftarrow \overline{b} \\
(A|B)\setminus c \\
\downarrow a \\
(A'|B)\setminus c \\
\ldots
\end{array}
\]

\[
\begin{array}{c}
(A'|B')\setminus c \\
\downarrow \overline{b} \\
(A'|B)\setminus c \\
\ldots
\end{array}
\]
Transition Graph

- Transition graph

\[
\begin{array}{c}
    \overline{b} \\
    (A|B) \setminus c \\
    a \\
    (A'|B) \setminus c \\
    b \\
\end{array}
\quad
\begin{array}{c}
    a \\
    (A|B') \setminus c \\
    \overline{b} \\
    (A'|B') \setminus c \\
    b \\
\end{array}
\]

- \((A|B) \setminus c = a.\tau.C\)
- \(C' := a.\overline{b}.\tau.C + \overline{b}.a.\tau.C\)

- Composite system
  - Behavior defined without use of composition combinator \mid or restriction combinator!

- Internal communication
  - \(\alpha.\tau.P = \alpha.P\)
  - \((A|B) \setminus c = a.D\)
  - \(D := a.\overline{b}.D + \overline{b}.a.D\)
The Basic Language

- Agent expressions
  - Agent constants and variables
  - Prefix $\alpha.E$
  - Summation $\sum E_i$
  - Composition $E_1 \mid E_2$
  - Restriction $E \backslash L$
  - Relabelling $E[f]$

- No value transmission between agents
  - Just synchronization.
The Transition Rules

- Act \( \alpha.E \xrightarrow{\alpha} E \)

- Sum \( \Sigma E_i \xrightarrow{\alpha} E'_j \)

- Com \( E \xrightarrow{\alpha} E' \)

- Com \( E\mid F \xrightarrow{\alpha} E\mid F' \)

- Com \( E \xrightarrow{l} E' \xrightarrow{T} F' \)

- Res \( E \xrightarrow{\alpha} E' \)

- Rel \( E [f] \xrightarrow{f(\alpha)} E'[f] \)

- Con \( P \xrightarrow{\alpha} P' \)
  \( A \xrightarrow{\alpha} P' \) (\( A := P \))
Derivatives and Derivation Trees

- **Immediate derivative of** $E$
  - Pair $(\alpha, E')$
  - $E \overset{\alpha}{\rightarrow} E'$
  - $E'$ is $\alpha$-derivative of $E$

- **Derivative of** $E$
  - Pair $(\alpha_1 \ldots \alpha_n, E')$
  - $E \overset{\alpha_1}{\rightarrow} \ldots \overset{\alpha_n}{\rightarrow} E'$
  - $E'$ is $(\alpha_1 \ldots \alpha_n)$-derivative of $E$

- **Derivation tree of** $E$

```
                 E_{11} \ldots
                /   \  \  
               α_{11} E_{1}
       / \         /  
 α_{1} E_1 \quad α_{12} E_{12} \ \ldots
               / \       /  
 α_{2} E_2 \quad \ \ldots
```
Examples of Derivation Trees

- Partial derivation tree
  \[(E|F)\backslash a\]
  \[\xrightarrow{\tau}\]
  \[((a.E + b.0) | \pi.F)\backslash a\]
  \[\xrightarrow{b}\]
  \[(0| \pi.F)\backslash a\]

- \(a.X + b.Y\)
  \[X\]
  \[\xrightarrow{a}\]
  \[a.X + b.Y\]
  \[\xrightarrow{b}\]
  \[Y\]

- Behavioural equivalence
  - Two agent expressions are behaviourally equivalent if they yield the same total derivation trees
The Value-Passing Calculus

- Values passed between agents
  - Can be reduced to basic calculus.
  - \( C := \text{in}(x).C'(x) \)
    \[ C'(x) := \text{out}(x).C \]
  - \( C := \Sigma_v \text{in}_v.C'_v \)
    \[ C'_v := \text{out}_v.C \quad (v \in V) \]
  - Families of ports and agents.

- The full language
  - Prefixes \( a(x).E, \overline{a}(e).E, \tau.E \)
  - Conditional if \( b \) then \( E \)

- Translation
  - \( a(x).E \Rightarrow \Sigma_v.E\{v/x\} \)
  - \( \overline{a}(e).E \Rightarrow \overline{a}_e.E \)
  - \( \tau.E \Rightarrow \tau.E \)
  - if \( b \) then \( E \Rightarrow \) (\( E \), if \( b \) and 0, otherwise)