This lecture is based on a number of different resources as indicated per slide.
Concurrent Programming

What is concurrency?

What makes concurrent programming different from sequential programming?

What are the core components of a concurrent language?
Concurrency

• Possible inter-thread communication mechanisms:
  • Read/write to shared memory.
  • Locks.
  • Monitors (a.k.a. wait/notify).
  • Buffered streams.
  • Unbuffered streams.
  • ...

• Which of these does Java support?
• Which should we include in a foundational calculus?
History

• Models of concurrency (late 1970s-80s): Communicating Sequential Processes (Hoare), Petri Nets (Petri), Calculus of Communicating Systems (Milner), ...

• Additional features to model dynamic network topologies (late 1980s-90s): Pi-calculus (Milner), Higher order pi-calculus (Sangiorgi), Ambients (Cardelli and Gordon), ...
In need of designated calculi
Program meanings

Program Meanings = Memories → Memories.

Program Meanings = Memories → P(Memories)

Ok for sequential programs

Needed for non-deterministic programs

© Ralf Lämmel, 2009-2012 unless noted otherwise
Parallelism and shared memory

Program $P_1 : x := 1 ; x := x + 1$
Program $P_2 : x := 2$

Semantics($P_1$) = Semantics($P_2$)
Parallelism and shared memory

Program $P_1 : x := 1 ; x := x + 1$
Program $P_2 : x := 2$
Program $Q : x := 3$

Program $R_1 : P_1 \text{ par } Q$
Program $R_2 : P_2 \text{ par } Q$

$\text{Semantics}(R_1) \neq \text{Semantics}(R_2)$
“Once the memory is no longer at the behest of a single master, then the master-to-slave (or: function-to-value) view of the program-to-memory relationship becomes a bit of a fiction. An old proverb states: He who serves two masters serves none. It is better to develop a general model of interactive systems in which the program-to-memory interaction is just a special case of interaction among peers.”
The shared memory model

Passive “thing”  Active process
Memory as an interactive process

Program variables as channels

Process

Process
Memory as a distributed process

Memory cells are processes.

Memories are no longer monolithic.
The Calculus of Communicating Systems
Agents and ports

• **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 $\overline{\text{out}}$. 

These slides were obtained by copy&paste&edit from W. Schreiner’s concurrency lectures (Kepler University, Linz).
A simple example

Behavior of $C$:

- $C := \text{in}(x).C'(x)$
- $C'(x) := \overline{\text{out}}(x).C$
Example: bounded buffers

Bounded buffer $\text{Buff}_n(s)$

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

• 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 $P$.

Later we add “composition”.

Process definitions may be parameterized.
Example: a vending machine

- Big chocolate costs 2p, small one costs 1p.
- \( V := 2p \cdot \text{big}.\text{collect}.V + 1p \cdot \text{little}.\text{collect}.V \)

Exercises:
Identify input vs. output.
What behaviors make sense for users?
Example: a multiplier

- \( \text{Twice} := \text{in}(x).\text{out}(2 \times x).\text{Twice} \).
- Output actions may take expressions.
Example: The JobShop

- A simple production line:
  - Two people (the jobbers).
  - Two tools (hammer and mallet).
  - Jobs arrive sequentially on a belt to be processed.

- Ports may be linked to multiple ports.
  - 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.
Example: The JobShop

- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - Jobs arrive sequentially on a belt to be processed.

- Ports may be linked to multiple ports.
  - 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.
The tools of the JobShop

- **Behaviors:**
  - **Hammer** := geth.Busyhammer
    
    *Busyhammer* := puth.Hammer
  - **Mallet** := getm.Busymallet
    
    *Busymallet* := putm.Mallet

- **Sort** = set of labels
  - $P : L \ldots$ agent $P$ has sort $L$
  - **Hammer**: \{geth, puth\}
    
    **Mallet**: \{getm, putm\}
  
    **Jobshop**: \{in, out\}
The jobbers of the JobShop

- Different kinds of jobs:
  - Easy jobs done with hands.
  - Hard jobs done with hammer.
  - Other jobs done with hammer or mallet.

- Behavior:
  - \( \text{Jobber} := \text{in}(\text{job}).\text{Start}(\text{job}) \)
  - \( \text{Start}(\text{job}) := \text{if} \ \text{easy}(\text{job}) \ \text{then} \ \text{Finish}(\text{job}) \)
    \( \text{else if} \ \text{hard}(\text{job}) \ \text{then} \ \text{Uhammer}(\text{job}) \)
    \( \text{else} \ \text{Usetool}(\text{job}) \)
  - \( \text{Usetool}(\text{job}) := \text{Uhammer}(\text{job}) + \text{Umallet}(\text{job}) \)
  - \( \text{Uhammer}(\text{job}) := \text{geth} . \text{puth} . \text{Finish}(\text{job}) \)
  - \( \text{Umallet}(\text{job}) := \text{getm} . \text{putm} . \text{Finish}(\text{job}) \)
  - \( \text{Finish}(\text{job}) := \text{out}(\text{done}(\text{job})) . \text{Jobber} \)
Composition of the 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 \( L \)
  - \( L \) internalizes all ports \( L \).
  - \((\text{Jobber} \mid \text{Jobber} \mid \text{Hammer})\)\{geth,puth\}

- **Complete system**:
  - \( \text{Jobshop} := (\text{Jobber} \mid \text{Jobber} \mid \text{Hammer} \mid \text{Mallet})\)\( L \)
  - \( L := \{\text{geth,puth,getm,putm}\} \)
"... sequential composition is indeed a special case of parallel composition ... in which the only interaction between occurs when $P$ finishes and $Q$ begins ..."

$P; Q$ not part of CCS

$P|Q$ part of CCS
Reformulations

- **Relabelling Operator**
  
  - $P[l'_1/l_1, \ldots, l'_n/l_n]$
  - $f(l) = \overline{f(l)}$

- **Semaphore agent**
  
  - $Sem := get.put.Sem$

- **Reformulation of tools**
  
  - $Hammer := Sem[geth/get, puth/put]$
  - $Mallet := Sem[getm/get, putm/put]$
In need of equality of agents

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

- **Claim:**
  - \( \text{Jobshop} = \text{Strongjobber} \mid \text{Strongjobber} \)
  - Specification of system \( \text{Jobshop} \)
  - Proof of equality required.

*In which sense are the processes equal?*
The core calculus
No value transmission: just synchronization

- Names and co-names
  - Set \( A \) of names (geth, ackin, ...)
  - Set \( A \) of co-names (geth, ackin, ...)
  - Set of labels \( L = A \cup \overline{A} \)

- Actions
  - Completed (perfect) action \( \tau \).
  - \( \text{Act} = L \cup \{\tau\} \)

- Transition \( P \xrightarrow{\ell} Q \) with action \( \ell \)
  - Hammer \( \xrightarrow{\text{geth}} \) Busyhammer
Transition rules of the core calculus

- **Act** \( \alpha . E \xrightarrow{\alpha} E \)
- **Sum** \( \sum E_i \xrightarrow{\alpha} E'_i \)
- **Com\(_1\)** \( E \xrightarrow{\alpha} E' \)
  \[ E_F \xrightarrow{\alpha} E'_F \]
- **Com\(_2\)** \( F \xrightarrow{\alpha} F' \)
  \[ E_F \xrightarrow{\alpha} E'_F \]
- **Com\(_3\)** \( E \xrightarrow{l} E' \)
  \[ F \xrightarrow{T} F' \]
  \[ E_F \xrightarrow{T} E'_F \]

- **Res** \( E \xrightarrow{\alpha} E' \)
  \[ E \backslash L \xrightarrow{\alpha} E' \backslash L \]  \( (\alpha, \overline{\alpha} \text{ not in } L) \)
- **Rel** \( E \xrightarrow{\alpha} E' \)
  \[ E[f] \xrightarrow{\text{f}(\alpha)} E'[f] \]
- **Con** \( P \xrightarrow{\alpha} P' \)
  \( A := P \)
The value-passing calculus

• Values passed between agents
  – Can be reduced to basic calculus.
  – $C := \text{in}(x).C''(x)$
    $C''(x) := \text{out}(x).C$
  – $C := \Sigma_v \text{in}_v.C'_v$
    $C'_v := \text{out}_v.C \ (v \in V)$
    – 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)
Bisimulation
(very informally)

• Two agent expressions \( P, Q \) are bisimular:
  • If \( P \) can do an \( \alpha \) action towards \( P' \),
  • then \( Q \) can do an \( \alpha \) action towards \( Q' \),
  • such that \( P' \) and \( Q' \) are again bisimular,
  • and v.v.

Intuitively two systems are bisimilar if they match each other's moves. In this sense, each of the systems cannot be distinguished from the other by an observer. [Wikipedia]
Laws
Summation laws

\[- P + Q = Q + P \]
\[- P + (Q + R) = (P + Q) + R \]
\[- P + P = P \]
\[- P + 0 = P \]
• Composition laws
  - $P|Q = Q|P$
  - $P|(Q|R) = (P|Q)|R$
  - $P|0 = P$

• Restriction laws
  - $P\setminus L = P$, if $L(P) \cap (L \cup \overline{L}) = \emptyset$.
  - $P\setminus K\setminus L = P\setminus(K \cup L)$
  - ...

• Relabelling laws
  - $P[id] = P$
  - $P[f][f'] = P[f' \circ f]$
  - ...

These slides were obtained by copy&paste&edit from W. Schreiner’s concurrency lectures (Kepler University, Linz).
Non-laws

• $\tau.P = P$
  
  - $A = a.A + \tau.b.A$
  
  - $A' = a.A' + b.A'$
  
  - $A$ may switch to state in which only $b$ is possible.
  
  - $A'$ always allows $a$ or $b$.

• $\alpha.(P + Q) = \alpha.P + \alpha.Q$
  
  
  - $b.P$ is $a$-derivative of right side, not capable of $c$ action.
  
  - $a$-derivative of left side is capable of $c$ action!
  
  - Action sequence $a, c$ may yield deadlock for right side.
• **Summary:** Calculus of Communicating Systems
  ✦ Modeling systems of interacting processes using channels.
  ✦ Approach amenable to formal analysis.
  ✦ Equivalence is based on communication behavior.

• **Recommended reading:**
  ✦ Milner’s “Elements of Interaction”
  ✦ CCS tutorial [AcetoLI05]