The Calculus of Communicating Systems

Ralf Lämmel
The Calculus of Communicating Systems (CCS)

- **Description of process networks**
  - Static communication topologies.

- **History sketch**
  - Robin Milner, 1980.
  - CCS: Calculus of Communicating Systems.
  - Various revisions and elaborations.
  - Later extended to mobile processes ($\pi$-calculus).

- **Algebraic approach**
  - Concurrent system modeled by term.
  - Theory of term manipulations.
  - Externally visible behavior preserved.

- **Observation 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 $\text{out}$. 
A simple example

Behavior of $C$:

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

Process behaviors are described as (mutually recursive) equations.
Behavior descriptions -- summary

- 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 as described by `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.
Another example: bounded buffers

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

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

Process definitions may be parameterized.
Example: a vending machine

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

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

- \textit{Twice} := \text{in}(x).\text{out}(2 \ast 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.

These slides were obtained by copy&paste&edit from W. Schreiner’s concurrency lectures (Kepler University, Linz).
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:
  - $\text{Hammer} := \text{geth}.\text{Busyhammer}$
  - $\text{Busyhammer} := \text{puth}.\text{Hammer}$
  - $\text{Mallet} := \text{getm}.\text{Busymallet}$
  - $\text{Busymallet} := \text{putm}.\text{Mallet}$

- Sort = set of labels
  - $P : L$ ... agent $P$ has sort $L$
  - $\text{Hammer} : \{\text{geth}, \text{puth}\}$
    - $\text{Mallet} : \{\text{getm}, \text{putm}\}$
    - $\text{Jobshop} : \{\text{in}, \text{out}\}$
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:
  - \(\text{Jobber} := \text{in}(\text{job}).\text{Start}(\text{job})\)
  - \(\text{Start}(\text{job}) := \text{if easy}(\text{job}) \text{then} \text{Finish}(\text{job}) \text{else if hard}(\text{job}) \text{then} \text{Uhammer}(\text{job}) \text{else Usetool}(\text{job})\)
  - \(\text{Usetool}(\text{job}) := \text{Uhammer}(\text{job}) + \text{Umallet}(\text{job})\)
  - \(\text{Uhammer}(\text{job}) := \text{geth}.\text{puth}.\text{Finish}(\text{job})\)
  - \(\text{Umallet}(\text{job}) := \text{getm}.\text{putm}.\text{Finish}(\text{job})\)
  - \(\text{Finish}(\text{job}) := \text{out}(\text{done}(\text{job})).\text{Jobber}\)
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 $L$.
  - `(Jobber | Jobber | Hammer)`\{geth,puth}\}

- **Complete system**:
  - `Jobshop := ` `(Jobber | Jobber | Hammer | Mallet)` \|$L$
  - `$L := \{\text{geth,puth, getm, putm}\}$

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

- **Relabelling Operator**
  - \( P[l_1'/l_1, \ldots, l_n'/l_n] \)
  - \( f(l) = f(l) \)

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

- **Reformulation of tools**
  - \( Hammer := Sem[\text{geth}/\text{get}, \text{puth}/\text{put}] \)
  - \( Mallet := Sem[\text{getm}/\text{get}, \text{putm}/\text{put}] \)
In need of equality of agents

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

- **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

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

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

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

- Transition $P \xrightarrow{l} Q$ with action $l$
  - Hammer $\text{geth} \rightarrow$ Busyhammer
Transition rules of the core calculus

- **Act** \( \alpha.E \xrightarrow{\alpha} E \)
- **Sum** \( j \in \sum \frac{E_j \xrightarrow{\alpha} E'_j}{\sum E_i \xrightarrow{\alpha} E'_j} \)
- **Com** \( 1 \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \)
- **Com** \( 2 \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E'|F'} \)
- **Com** \( 3 \frac{E \xrightarrow{l} E' F \xrightarrow{\tau} F'}{E|F \xrightarrow{\tau} E'|F'} \)
- **Res** \( \frac{E \xrightarrow{\alpha} E'}{E\setminus L \xrightarrow{\alpha} E'\setminus L} \)
- **Rel** \( \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{\alpha} E'[f]} \)
- **Con** \( \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \)

This rule makes clear that no more than two agents participate in communication.

This rule rules out transitions with hidden names.

This is about the application of definitions for agents.
The value-passing calculus

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

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

- Translation
  - $a(x).E \Rightarrow \sum_v.E\{v/x\}$
  - $\overline{a}(e).E \Rightarrow \overline{a}_e.E$
  - $\tau.E \Rightarrow \tau.E$
  - $\text{if } b \text{ then } E \Rightarrow (E, \text{if } b \text{ and } 0, \text{otherwise})$
Derivation trees
(Exhaustive application of the transition relation)

• Derivation tree of $E$

\[
\begin{array}{c}
E \\
E_1 \\
E_{11} \ldots \\
\vdots
\end{array}
\]

\[
\begin{array}{c}
E \\
E_1 \\
E_{11} \ldots \\
\vdots
\end{array}
\]

\[
\begin{array}{c}
E \\
E_1 \\
E_{11} \ldots \\
\vdots
\end{array}
\]

Behavioral equivalence: two agent expressions are behaviorally equivalent if they yield the same total derivation trees.
From infinite derivation trees ...

\[(A | B) \downarrow c \quad \downarrow a \quad (A' | B) \downarrow c \quad \downarrow \tau \quad (A | B') \downarrow c \quad \downarrow b\]

\[\begin{align*}
- & A := a.A', \quad A' := \overline{c}.A \\
- & B := c.B', \quad B' := \overline{b}.B
\end{align*}\]
... to finite transition graphs

- \( A := a.A', A' := \overline{c}.A \)
- \( B := c.B', B' := \overline{b}.B \)

\[
\begin{array}{c}
\overline{b} \\
\rightarrow \\
(A'|B')\backslash c \\
\downarrow \\
\rightarrow \\
(A\backslash B)\backslash c \\
\downarrow \\
\rightarrow \\
A\backslash B \\
\downarrow \\
\rightarrow \\
(A\backslash B)\backslash c
\end{array}
\]

- \( (A\backslash B)\backslash c \) b-equivalent to \( a.\tau.C \)
- \( C := a.\overline{b}.\tau.C + \overline{b}.a.\tau.C \)

Behavior can be defined by \( + \) and \( . \) only!
Internal versus external actions

- **Action** \( \tau \):
  - Simultaneous action of both agents.
  - *Internal* to composed agent.

- **Internal actions should be ignored.**
  - Only external actions are visible.
  - Two systems are *observationally equivalent* if they exhibit same pattern of external actions.

- \( P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n \) o-equivalent to \( P \xrightarrow{\tau} P_n \)

- \( \alpha.\tau.P \) o-equivalent to \( \alpha.P \)

- **Simpler variant of** \((A|B)\setminus c\):
  - \((A|B)\setminus c\) o-equivalent to \( a.D \)
  - \( D := a.\overline{b}.D + \overline{b}.a.D \)

Internal actions take no “time”.
In need of bisimulation

- Example agents $A$ and $B$
  - $A = a.(b.0 + c.d.A)$
  - $B = a.b.0 + a.c.d.B$

- “Language understood” by $A$ and $B$
  - $(a.c.d)^* . a.b.0$
  - $A$ and $B$ seem equivalent.

- Ports $a$, $b$, $c$, $d$.
  - Initially only $a$ is “unlocked”.
  - Observer “presses button” $a$.
  - In $A$, $b$ and $c$ are “unlocked”.
  - In $B$, sometimes $b$, sometimes $c$ is “unlocked”.
  - $A$ and $B$ can be experimentally distinguished!

Think of repeatedly “replaying” the system from the state that was obtained by pressing $a$. 
Bisimulation (very informally)

- Two agent expressions $P$, $Q$ are bisimilar:
  - 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 bisimilar,
  - 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**
  
  \[ P\backslash L = P, \text{ if } L(P) \cap (L \cup L') = \emptyset. \]
  \[ P\backslash K \backslash L = P\backslash (K \cup L) \]
  \[ \ldots \]

• **Relabelling laws**
  
  \[ P[\text{id}] = P \]
  \[ P[f][f'] = P[f' \circ f] \]
  \[ \ldots \]
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:** CCS

  ♦ An algebraic approach to system modeling.
  ♦ Approach amenable to formal analysis.
  ♦ Equivalence is based on communication behavior.

• **Prepping:** Read CCS tutorial [AcetoLI05]

• **Lab:** Model CCS in Prolog

• **Outlook:**

  ♦ Ending of Prolog section
  ♦ Beginning of Haskell section
  ♦ Midterm