CCS

- Introduction
- Modeling Communication
- The Basic Language

What is CCS?

CCS is a theory of communicating systems that reveals the essence of concurrency and communication.

It can also be considered as a formal notation to model and analyze the behavior of concurrent systems.

Why CCS?

- Offers deep insights about concurrency and communication
- Helps to distinguish essential differences from accidental ones
- Provides a mathematic treatment that highlights precise modeling and analysis

CCS vs Net Theory

Net theory is a generalization of the theory of automata that allows for the occurrence of independent actions.

CCS is based on the notion of observational equivalence, while net theory is concerned with the causality of the actions.

CCS

- Introduction
- Modeling Communication
- The Basic Language

Agent

A complex system is often composed of several parts that communicate with each other. Each of the parts has its own identity and is called an agent.

The term agent will be used broadly: For one purpose, we take it to be atomic; for other purposes, we take it as a composition of sub-agents.

Action

Each action of an agent is either an interaction with its neighboring agents, and then it is communication, or it occurs independently of them and then it may occur concurrently with their actions.

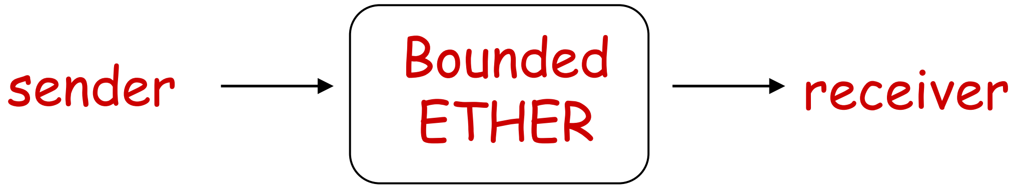
A central idea in CCS is observational equivalence: Two agents are equivalent if they exhibit the same behavior in terms of observable actions.

Medium - Ether

sender → ETHER → receiver

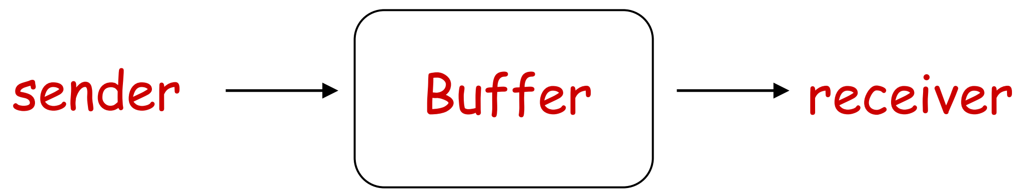
- The sender may always send a message
- The receiver may always receive a message, provided that a medium is not empty
- The order of messages may not be preserved.

Medium - Bounded Ether



- The sender may always send a message, provided the medium is not full.
- The receiver may always receive a message, provided that a medium is not empty
- The order of messages may not be preserved.

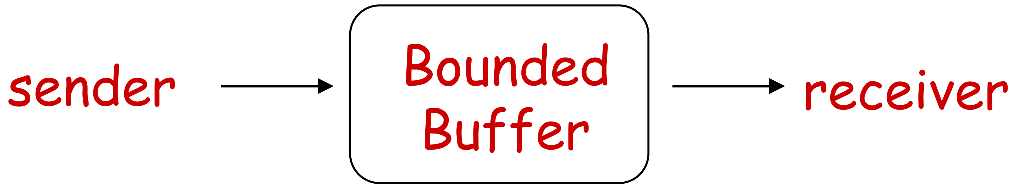
Medium – Buffer



- The sender may always send a message.
- The receiver may always receive a message, provided that a medium is not empty
- The order of messages is preserved.

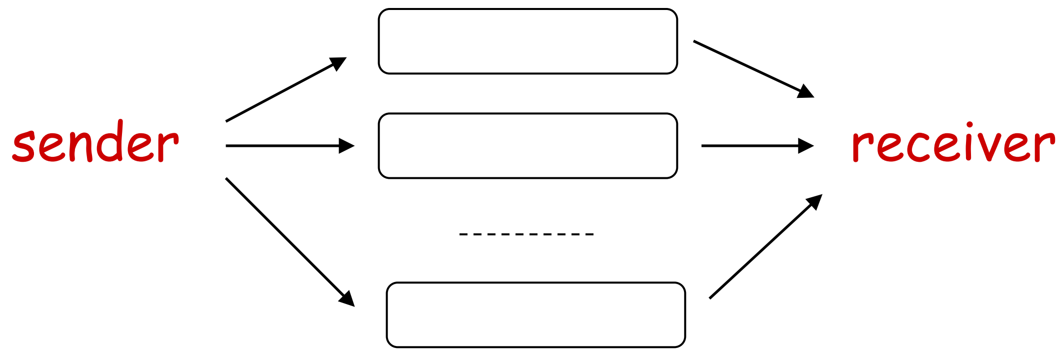
UTA

Medium – Bounded Buffer



- The sender may always send a message, provided the medium is not full.
- The receiver may always receive a message, provided the medium is not empty.
- The order of messages is preserved.

Medium - Shared Memory



- The sender may always write an item to a memory location.
- The receiver may always read an item from a memory location.
- Writing and reading may occur in any order.

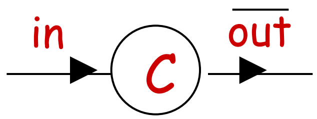
Handshake

Each arrow in each diagram is a single action, called handshake, that is indivisible in time and consists of the passage of a piece of information between two entities.

We consider that medium is no different from sender/receiver, in the sense that they are participants of a handshake.

sender $\longrightarrow$ receiver

Agent Expression



$$\Rightarrow \quad C \stackrel{def}{=} in(x).C'(x) \qquad C'(x) \stackrel{def}{=} \overline{out}(x).C \quad \Rightarrow \quad C \stackrel{def}{=} in(x).\overline{out}(x).C$$

- C is an agent that may hold a single data item. It can accept an item at port in, and deliver an item at port out.
- in(x) stands for a handshake in which an item is received at in, and out(x) stands for a handshake in which an item is delivered at out.

Scope

The scope of a variable on the left-hand side of a defining equation is the entire equation.

The scope of a variable in a prefix on the right-hand side is the agent expression which begins with the prefix.

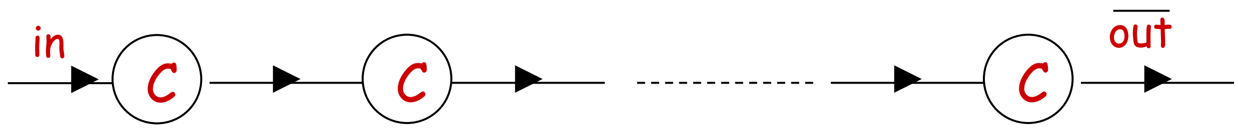
Note that a variable never has scope larger than a single equation.

Abstraction

An agent can be specified at different levels of abstraction – either directly in terms of its interactions with the environment, or indirectly in terms of its composition from smaller agents.

One important power provided by CCS is to prove that agents specified at different levels of abstractions are equivalent.

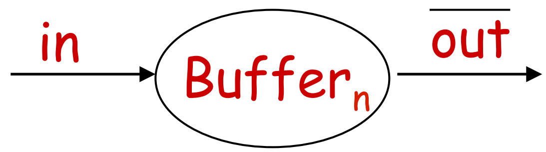
N-cell Buffer (1)



$$C^{(n)} \overset{def}{=} C \wedge C \wedge \ldots \wedge C$$

$C \wedge C$ represents an agent formed by linking the out port of the first sub-agent C with the in port of the second sub-agent C.

N-cell Buffer (2)



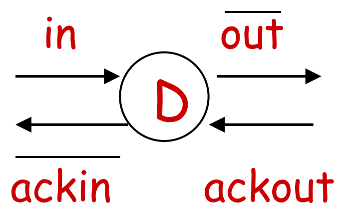
$$Buff_n(\varepsilon) \stackrel{def}{=} in(x).Buff_n(x)$$

$$Buff_n(s{:}v) \stackrel{def}{=} \overline{out}(v).Buff_n(s) \quad (|s| = n - 1)$$

$$Buff_n(s{:}v) \stackrel{def}{=} \begin{array}{l} in(x).Buff_n(x : s : v) + \\ \overline{out}(v).Buff_n(s) \quad (|s| < n - 1) \end{array}$$

- $P + Q$ represents an agent that behaves like P or Q; as soon as one performs its first action, the other is discarded.
- ε - empty sequence; $:$ - sequence concatenation.

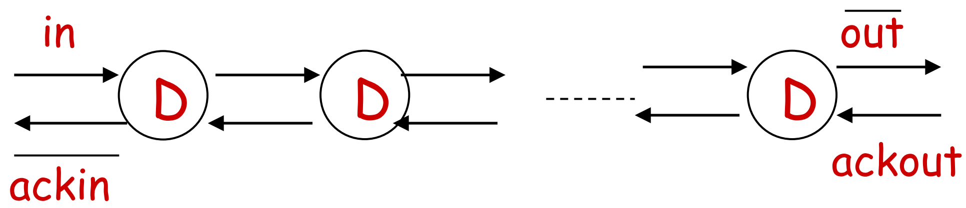
Buffers with acks (1)



$$D \stackrel{def}{=} in(x).\overline{out}(x).ackout.\overline{ackin}.D$$

D will acknowledge the receipt of an input value only after it has delivered the value as output and also received acknowledgement for it.

Buffer with acks (2)



$$D^{(n)} \stackrel{def}{=} D \,\hat{}\, D \,\hat{}\, \ldots \,\hat{}\, D$$

Question: What is the capacity of $D^{(n)}$?

Semaphore (1)

$$Sem_n(0) \stackrel{def}{=} get.Sem_n(1)$$

$$Sem_n(k) \stackrel{def}{=} get.Sem_n(k+1) + put.Sem_n(k-1)$$

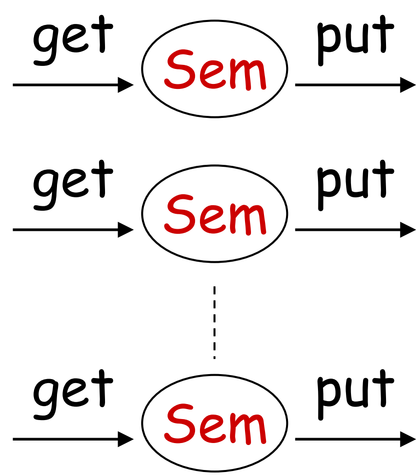
$$Sem_n(n) \stackrel{def}{=} put.Sem_n(n-1)$$

Note that Sem_n admits any sequence of gets and puts in which the number of gets minus the number of puts lies in the range of 0 to n inclusive.

Question: What is the difference between Sem_n and $Buff_n$?

Semaphore (2)

$$Sem^{(n)} \stackrel{def}{=} Sem \mid Sem \mid \ldots \mid Sem$$



Composition (|) : Agent P | Q is a system in which P and Q may proceed independently but may also interact through complementary ports.

CCS

- Introduction
- Modeling Communication
- The Basic Language

Synchronization

Synchronization is a special form of communication in which no data value was transmitted.

We will first focus on a basic calculus of pure synchronizations, which has no value variables and expressions.

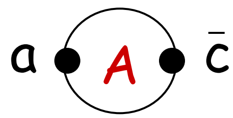
Later we will show that calculus involving value variables, expressions and conditions can be translated it into the basic calculus.

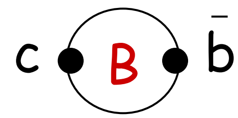
Transition

Each action transforms an agent from one state to another.

$$P \stackrel{def}{=} a.Q \quad \equiv \quad P \xrightarrow{a} Q$$

Composition (1)





$$A \stackrel{def}{=} a.A'$$

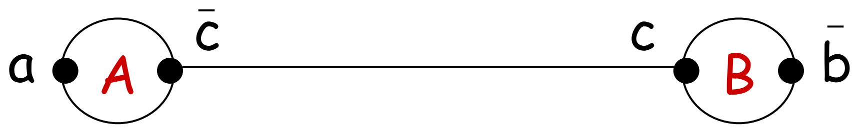
$$A' \stackrel{def}{=} \bar{c}.A$$

$$B \stackrel{def}{=} c.B'$$

$$B' \stackrel{def}{=} \bar{b}.B$$

What are the transition rules for agent $A \mid B$?

Composition (2)



$$A \xrightarrow{a} A' \quad \Rightarrow \quad A \mid B \xrightarrow{a} A' \mid B$$

$$A' \xrightarrow{\bar{c}} A \quad \Rightarrow \quad A' \mid B \xrightarrow{\bar{c}} A \mid B$$

$$B \xrightarrow{c} B' \quad \Rightarrow \quad A \mid B \xrightarrow{c} A \mid B'$$

$$B' \xrightarrow{\bar{b}} B \quad \Rightarrow \quad A \mid B' \xrightarrow{\bar{b}} A \mid B$$

$$\begin{array}{l} A' \xrightarrow{\bar{c}} A \\ B \xrightarrow{c} B' \end{array} \quad \Rightarrow \quad A' \mid B \xrightarrow{\tau} A \mid B'$$

Internal Action

An important feature of CCS is to ignore, as far as possible, internal actions when analyzing the behavior of composite systems.

The reason is that internal actions are not directly observable. Two systems will be regarded as equivalent if they exhibit the same pattern of external actions.

$$P \xrightarrow{\tau} P1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} Pn$$

$$||$$

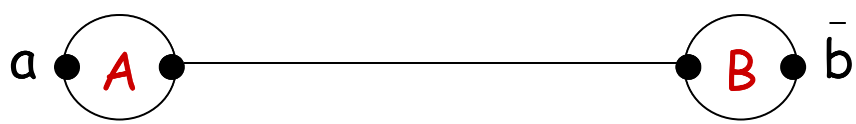
$$P \xrightarrow{\tau} Pn$$

Restriction

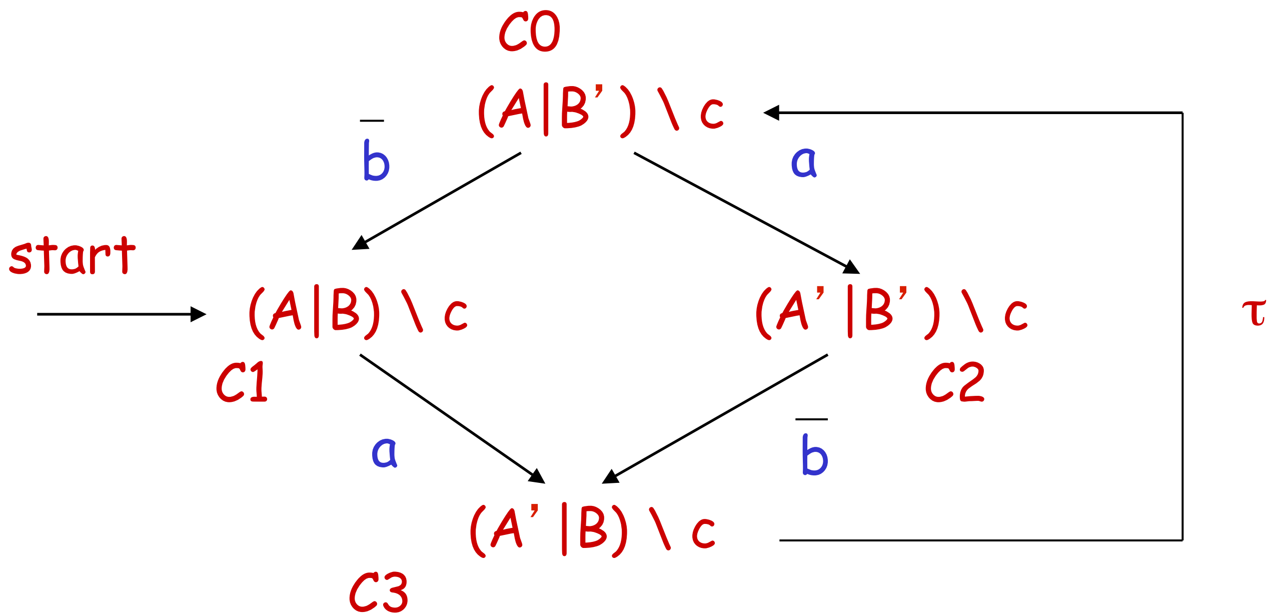
A composite system can be restricted from performing some actions.

$$P \xrightarrow{\alpha} Q \quad \Rightarrow \quad P \backslash L \xrightarrow{\alpha} Q \backslash L \quad \text{if } \alpha \text{ and } \bar{\alpha} \text{ are not in } L$$

For example, consider the composite system (A | B) \ c. What are the transition rules of this system?



Transition Graph



$$C_0 \stackrel{def}{=} \bar{b}.C_1 + a.C_2$$

$$C_1 \stackrel{def}{=} a.C_3$$

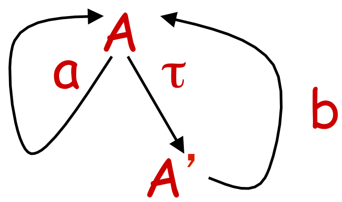
$$C_2 \stackrel{def}{=} \bar{b}.C_3$$

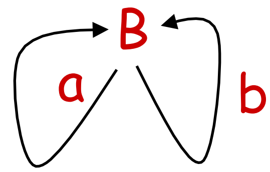
$$C_3 \stackrel{def}{=} \tau.C_0$$

$\tau.P = P$?

Consider the following two agents:

$A \stackrel{def}{=} a.A + \tau.b.A$ $\qquad\qquad$ $B \stackrel{def}{=} a.B + b.B$





UTA

The Basic Language (1)

- $A/\bar{A}$: An infinite set of names/co-names
- $\Phi = A \cup \bar{A}$, $Act = \Phi \cup \{\tau\}$
- a, b, c range over A, $\alpha, \beta, \ldots$ range over Act
- f: a relabelling function f is a function from Φ to Φ such that $f(\bar{l}) = \overline{f(l)}$, where $l \in \Phi$, and $f(\tau) = \tau$.

Today's Agenda

- HW 4
- Continue on CCS

The Basic Language (2)

Given a set X of agent variables and a set K of agents, the set Ω of agent expressions is the smallest set as defined below:

- $X \subseteq \Omega$, $K \subseteq \Omega$
- Let E and E_i be two expressions in Ω.
  - $\alpha.E$, a prefix ($\alpha \in Act$)
  - $\sum_{i\in I} E_i$, a summation, where I is an indexing set
  - $E_1 \mid E_2$, a composition
  - $E\backslash L$, a Restriction ($L \subseteq \Phi$)
  - $E[f]$, a relabelling (f: a relabelling function)

The Basic Language (3)

- $\sum_{i \in I} E_i$ is also written as $E_1 + E_2$ if $I = \{1, 2\}$.
- $\sum_{i \in I} E_i = 0$, if I is empty. Note that 0 is an inactive agent.
- Binding power: Restriction = Relabelling > Prefix > Composition > summation
- An agent expression is an agent if it contains no free variables. A Constant is an agent whose meaning is given by a defining equation.
- $E_1 \equiv E_2$: E_1 and E_2 are syntactically identical.
- $E_1 = E_2$: E_1 and E_2 are equivalent in terms of their behavior.

UTA

Transition Rules (1)

Act $$\frac{}{\alpha.E \xrightarrow{\alpha} E}$$

Com$_1$ $$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$$

Com$_2$ $$\frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$

Sum$_j$ $$\frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j}$$

Com$_3$ $$\frac{E \xrightarrow{\alpha} E' \quad F \xrightarrow{\bar{\alpha}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

Transition Rules (2)

Res $$\frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L} \quad \alpha, \overline{\alpha} \notin L$$

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

Con $$\frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} A'} \quad (A \stackrel{def}{=} P)$$

UTA

Inference Diagram (1)

An inference diagram can be used to justify a transition of any agent expression.

For example, consider how to justify the following transition:

$$((a.E + b.0) \mid \overline{a}.F)\backslash a \xrightarrow{\tau} (E|F)\backslash a$$

Inference Diagram (2)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a.E \xrightarrow{a} E}\,\text{Act}}{a.E + b.0 \xrightarrow{a} E}\,\text{Sum}_1 \qquad \dfrac{}{\bar{a}.F \xrightarrow{\bar{a}} F}\,\text{Act}}{(a.E + b.0) \mid \bar{a}.F \xrightarrow{\tau} E|F}\,\text{Com}_3}{((a.E + b.0) \mid \bar{a}.F)\backslash a \xrightarrow{\tau} (E|F)\backslash a}\,\text{Res}$$

Inference Diagram (3)

$$(A|B)\backslash c \xrightarrow{a} (A'|B)\backslash c$$

$$\text{Act}\ \frac{}{a.A' \xrightarrow{a} A'}$$

$$\text{Con}\ \frac{}{A \xrightarrow{a} A'}$$

$$\text{Com}_1\ \frac{}{A|B \xrightarrow{a} A'|B}$$

$$\text{Res}\ \frac{}{(A|B)\backslash c \xrightarrow{a} (A'|B)\backslash c}$$

UTA

Derivative

Whenever $E \xrightarrow{\alpha} E'$, we call the pair (α, E') an immediate derivative of E, we call α an action of E, and we call E' an α-derivative of E.

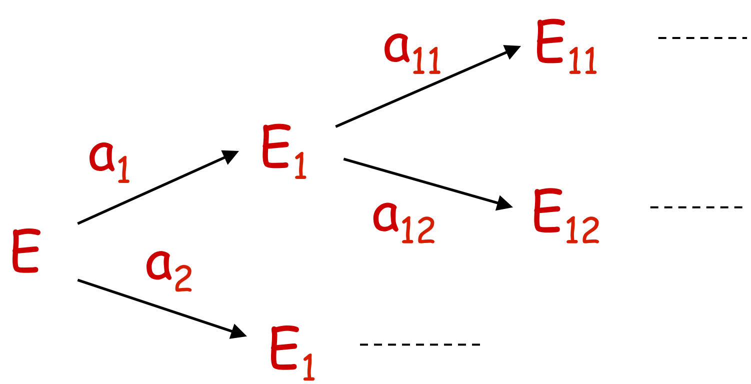
Whenever $E \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} E'$, we call $(\alpha_1 \ldots \alpha_n, E')$ a derivative of E, we call $\alpha_1 \ldots \alpha_n$ an action-sequence of E, and we call E' an $\alpha_1 \ldots \alpha_n$-derivative of E.

Note that if $n = 0$, we have that ε is an action-sequence of E, and E is a derivative of itself.

Derivative Tree (1)

A derivative tree of an agent expression E consists of all the derivatives of E.

A derivative tree is total if every terminal node has no immediate derivatives, otherwise partial.



Derivative Tree (2)

$$\begin{tikzcd}
 & (E \mid F)\backslash a \\
((a.E + b.0) \mid \bar{a}.F)\backslash a \arrow[ur, "\tau"] \arrow[dr, "b"] & \\
 & (0 \mid \bar{a}.F)\backslash a
\end{tikzcd}$$

The meaning of an agent should be a property of its derivation tree disregarding the expressions which lie at the nodes.

Sort

For any $L \subseteq \Phi$, if the actions of an agent and all its derivatives lie in $L \cup \{\tau\}$ then we say P has sort L, or L is a sort of P, and write $P : L$.

For every E and L, L is a sort of E if and only if whenever $E \xrightarrow{\alpha} E'$, then (1) $\alpha \in L \cup \{\tau\}$ (2) L is a sort of E'.

Syntactic Sort (1)

Given the sorts $L(A)$ and $L(X)$ of constants and variables, the syntactic sort $L(E)$ of an agent expression E is defined below:

- $L(a.E) = \{a\} \cup L(E)$
- $L(\tau.E) = L(E)$
- $L(\sum_{i \in I} E_i) = \cup_{i \in I} L(E_i)$
- $L(E|F) = L(E) \cup L(F)$
- $L(E \backslash L) = L(E) - (L \cup \overline{L})$
- $L(E[f]) = \{f(a): a \in L(E))$
- In addition, if $A \overset{def}{=} P$, then $L(P) \subseteq L(A)$ must hold.

UTA

Syntactic Sort (2)

In many cases, calculating L(E) involves little more than collecting up the labels which occur "free" in E, i.e., not bound by a restriction.

$$P \equiv ((a.0 + b.0) \mid (\overline{b}.0 + c.0)) \setminus b$$

What is the sort of P?

UTA

Value Passing

The full calculus with value passing can be reduced to the basic calculus.

$$C \stackrel{def}{=} in(x).C'(x)$$
$$C'(x) \stackrel{def}{=} \overline{out}(x).C$$

$$\Downarrow$$

$$C \stackrel{def}{=} \sum_{v \in V} in_v.C_v$$
$$C'_v \stackrel{def}{=} \overline{out}_v.C \quad (v \in V)$$