# Chapter 2

# A Calculus of Communicating Systems

Goals for Chapter 2:

- Syntax of CCS
  - symbols for actions and for agents,
  - rules for writing well-formed agent expressions.
- Semantics of CCS in terms of agents' behaviour.
  - transition trees, transition graphs and their use in representing the behaviour of systems.
  - inference trees as means to prove or disprove transitions of agent expressions.
- CCS as a formalism for the modelling (the behaviour of) concurrent systems.

**Value-passing vs pure synchronisation**

Is CCS with pure synchronisation, and without value-passing, adequate for the modelling of realistic concurrent systems?

- **YES** in theory
- **NO** in practice

First, we develop the basic calculus, where agent expressions have no value parameters.

Then, we define a larger calculus, called the value-passing calculus, in terms of the basic one.

# 1. Actions and Transitions

## Actions

We assume the following infinite sets:

- $\mathcal{A}$: the set of names of actions, or simply actions, ranged over by $a$, $b$, $c$,. . . , e.g.

  $$geth,\ puth$$

- $\overline{\mathcal{A}}$: the set of co-names, or co-actions, ranged over by $\overline{a}$, $\overline{b}$, $\overline{c}$, . . . , e.g.

  $$\overline{geth},\ \overline{puth}$$

- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$: the set of labels, ranged over by $l$, $l'$.
- Extend complementation to the whole of $\mathcal{L}$, so that $\overline{\overline{a}} = a$.
- $Act = \mathcal{L} \cup \{\tau\}$: ranged over by $\alpha, \beta$.
- In the basic calculus, labels have **no** value parameters.

## Our Objectives

- The behaviour of agents will be represented in terms of transitions.
- How do we find transitions of agents?
- Transition rules for CCS operators (constructs).
- How do we represent transitions of agents?
  - Transition trees.
  - Transition graphs.
- Inference trees as means to prove or disprove transitions of agents.

**Transitions**

The behaviour of an agent will be defined in terms of all its possible transitions. An agent can be thought of as an automaton:

- States are labelled with agent expressions.
- A transition from state $P$ to state $Q$ represents an action of agent $P$.
- We write such a transition as

  $$P \xrightarrow{l} Q$$

The following three statements mean the same

- $P \xrightarrow{l} Q$
- A transition from state $P$ to state $Q$ by $l$
- Agent $P$ performs an action $l$ and then behaves like (becomes) agent $Q$.

**How do we find transitions of CCS agents?**

We **infer** them from the structure of agent expressions using the **transition rules** for the constructs (operators) of CCS.

**Transition rules**

Transition rules are conditional statements which define transitions of agents in terms of (or depending upon) transitions of their immediate components.

Informally these rules have the form

**Since** the components have such and such transitions
**we infer** a transition of the agent
[ **provided** some condition holds ]

For example, consider the agent $A + B$.

A transition rule for the agent $A + B$, i.e. $A$ and $B$ composed with $+$, tells us how the behaviour of $A + B$ depends on the behaviour of $A$ and $B$ :

**Since** $A \stackrel{\alpha}{\rightarrow} A'$
**we infer** $A + B \stackrel{\alpha}{\rightarrow} A'$

This is an instance of one of the transition rules for $+$:

$$\frac{E \stackrel{\alpha}{\rightarrow} E'}{E + F \stackrel{\alpha}{\rightarrow} E'}$$

where $E, F$ are any agent expressions.

**Prefix**

The agent $Hammer$ was defined by

$$Hammer \stackrel{def}{=} geth.Busyhammer$$

$$Busyhammer \stackrel{def}{=} puth.Hammer$$

We notice the the agent can be in one of the two states: $Hammer$ and $Busyhammer$. Movements between these states are accompanied by actions $geth$ and $puth$, respectively. We guess

$$Hammer \stackrel{geth}{\rightarrow} Busyhammer$$
$$Busyhammer \stackrel{puth}{\rightarrow} Hammer$$

Since we use the Prefix combinator to express sequences of actions of agents, it should not be surprising that agents of the form $\alpha.E$ can perform $\alpha$, or have $\alpha$ transitions.

The transition rule defining the operational meaning of the Prefix combinator is

$$\frac{}{\alpha.E \stackrel{\alpha}{\rightarrow} E}$$

or equivalently, and more simply

$$\alpha.E \stackrel{\alpha}{\rightarrow} E$$

where $\alpha$ is any action and $E$ any agent.

Although the rule can be used to infer

$$puth.Hammer \xrightarrow{puth} Hammer$$

we still do not know how to infer transitions of $Busyhammer$.

**Agent Constants**

The definitions of agent constants such as, for example, $A$ and $Busyhammer$ have the form

$$A \stackrel{def}{=} P$$

where $P$ is any agent expression (with no variables).

By the above definition, the behaviour of $A$ is described by $P$, so we infer that the transitions of $A$ are precisely the transitions of $P$.

This is expressed formally as the transition rule for agent constant:

$$\frac{P \stackrel{\alpha}{\rightarrow} P'}{A \stackrel{\alpha}{\rightarrow} P'} \quad A \stackrel{def}{=} P$$

Since

$$Busyhammer \stackrel{def}{=} puth.Hammer$$

and

$$puth.Hammer \xrightarrow{puth} Hammer$$

we infer, using the rule above,

$$Busyhammer \xrightarrow{puth} Hammer.$$

**Parallel Composition**

**Independent actions**

$A$ can do **any** of its actions when composed in parallel with $B$ (written as $A|B$), leaving $B$ undisturbed:

Since $A \xrightarrow{\alpha} A'$
we infer $A|B \xrightarrow{\alpha} A'|B$

Correspondingly, for $B$ we have:

Since $B \xrightarrow{\alpha} B'$
we infer $A|B \xrightarrow{\alpha} A|B'$

Here, $a$ and $\overline{c}$ are any visible actions.

Note: the above statements are 'quantified' over all actions $\alpha$ (visible or silent) that agents $A$ and $B$, respectively, can perform.

**Communication**

A communication (handshake) changes the states of the participating (component) agents simultaneously, and results in action $\tau$. Here, $c$ is any visible action and it cannot be $\tau$:

Since $A \xrightarrow{\overline{c}} A'$ and $B \xrightarrow{c} B'$
we infer $A|B \xrightarrow{\tau} A'|B'$

This is formally written as transition rule

$$\frac{A \xrightarrow{\overline{c}} A' \quad B \xrightarrow{c} B'}{A|B \xrightarrow{\tau} A'|B'}$$

**Features of silent actions**

- They are perfect (or completed) actions.
- They do not represent a potential for communication.
- They are not observable, i.e. cannot be communicated upon by agents or restricted.
- Only external actions of agents are observable, i.e. important as far as our understanding of systems' behaviour is concerned.

We use a single symbol $\tau$ to represent **all** such handshakes.

Let **the set** of actions be

$$Act = \mathcal{L} \cup \{\tau\}$$

Let $\alpha$ and $\beta$ range over $Act$.

**The nondeterministic choice**

Agent $A + B$ behaves either like $A$ or $B$. This is described by the following rules, where $\alpha$ is any action, visible or silent.

$$\frac{A \stackrel{\alpha}{\rightarrow} A'}{A + B \stackrel{\alpha}{\rightarrow} A'} \qquad \frac{B \stackrel{\alpha}{\rightarrow} B'}{A + B \stackrel{\alpha}{\rightarrow} B'}$$

Although this seems simple, mixing $+$ with actions $\tau$ sometimes produces unexpected results.

Are these agent equivalent?

$$a.A + \tau.b.A \quad \text{and} \quad a.A + b.A$$

**Restriction**


$$(A|B)\backslash c$$

- $(A|B)\backslash c$ cannot perform $c$ or $\overline{c}$
- But it can perform $\tau$ which results from the communication on $c$.

**Transition rule for restriction**

| | |
|---|---|
| **Since** | $E \xrightarrow{\alpha} E'$ |
| **we infer** | $E\backslash L \xrightarrow{\alpha} E'\backslash L$ |
| **provided** | $\alpha, \overline{\alpha} \notin L$ |

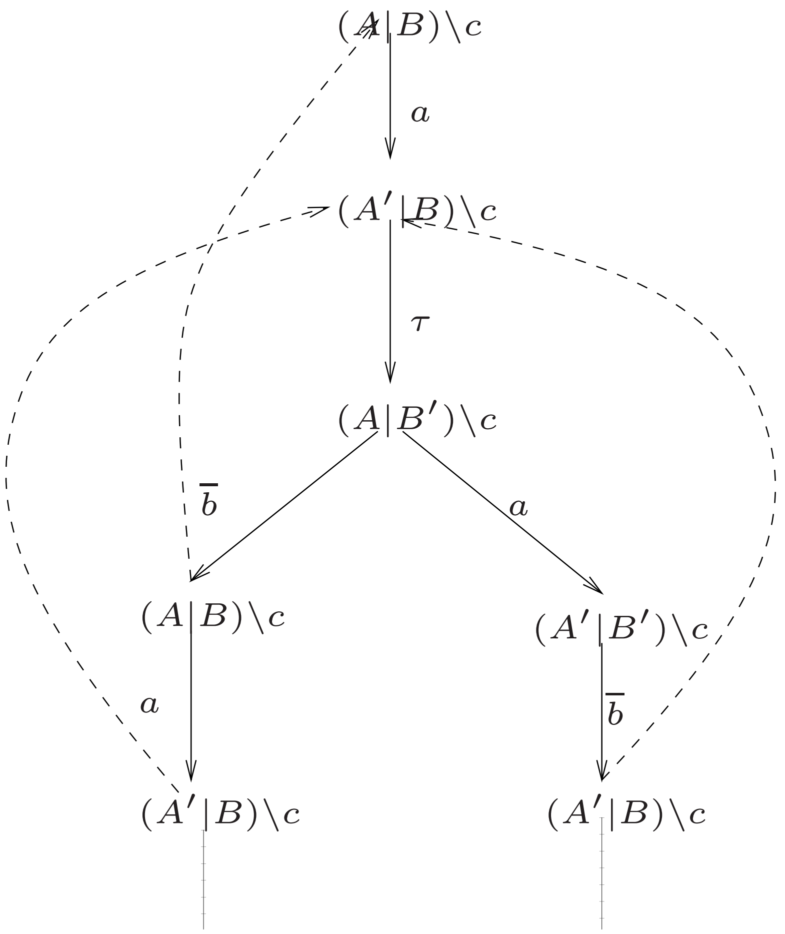
This is also written as

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

**Transition trees**

$$A \stackrel{def}{=} a.A' \qquad A' \stackrel{def}{=} \overline{c}.A$$

$$B \stackrel{def}{=} c.B' \qquad B' \stackrel{def}{=} \overline{b}.B$$



**Transition graphs**

With $A$ and $B$ defined as above, namely

$$\begin{array}{ll} A \stackrel{def}{=} a.A' & \quad A' \stackrel{def}{=} \overline{c}.A \\ B \stackrel{def}{=} c.B' & \quad B' \stackrel{def}{=} \overline{b}.B \end{array}$$

we can fold the transition tree of $A|B$ into the following transition graph:

$$\begin{tikzcd}
 & (A|B')\backslash c \arrow[dl, "\overline{b}"'] \arrow[dr, "a"] & \\
(A|B)\backslash c \arrow[dr, "a"'] & & (A'|B')\backslash c \arrow[dl, "\overline{b}"] \\
 & (A'|B)\backslash c \arrow[uu, "\tau"'] &
\end{tikzcd}$$

Is the following graph different from the previous one?

$$\begin{tikzcd}
& C_0 \arrow[dl, "\overline{b}"'] \arrow[dr, "a"] & \\
C_1 \arrow[dr, "a"'] & & C_2 \arrow[dl, "\overline{b}"] \\
& C_3 \arrow[uu, "\tau"'] &
\end{tikzcd}$$

It is not different, providing that we replace $(A|B')\backslash c$, $(A|B)\backslash c$, $(A'|B')\backslash c$, and $(A'|B)\backslash c$, by $C_0$, $C_1$, $C_2$, and $C_3$ respectively.

We can guess these equations by inspecting the graph:

$$\begin{array}{ll} C_0 \stackrel{def}{=} \overline{b}.C_1 + a.C_2 & \qquad C_1 \stackrel{def}{=} a.C_3 \\ C_2 \stackrel{def}{=} \overline{b}.C_3 & \qquad C_3 \stackrel{def}{=} \tau.C_0 \end{array}$$

Then, we claim

$$(A|B)\backslash c = C_1$$

Alternatively, $(A|B)\backslash c = a.\tau.C_0$, where

$$C_0 \stackrel{def}{=} a.\overline{b}.\tau.C_0 + \overline{b}.a.\tau.C_0$$

Further, we should be able to ignore (some, all?) $\tau$ actions to obtain

$$(A|B)\backslash c = a.C$$

where

$$C \stackrel{def}{=} a.\overline{b}.C + \overline{b}.a.C.$$

## 2. The Pre-emptive Power of $\tau$

Actions $\tau$ are not observable.

So, can we simplify agent expressions by removing some or all of silent actions from agent expressions?

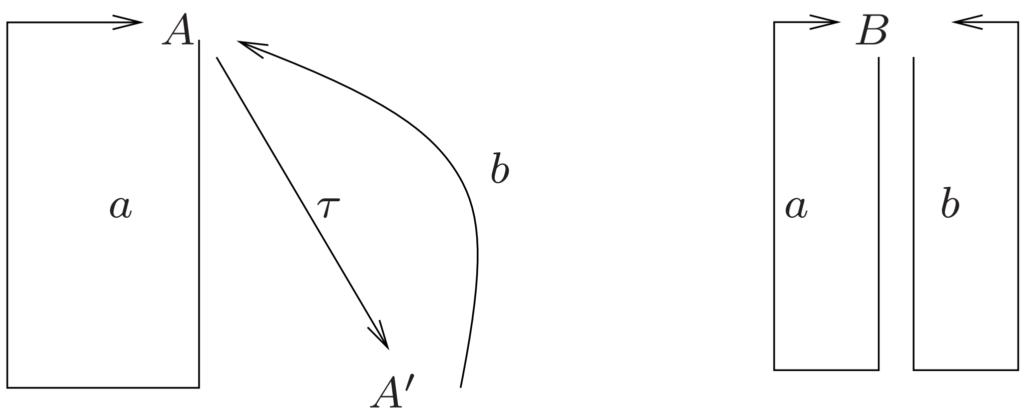
Which of the following should we accept as valid?

$$(1) \quad P = \tau.P \qquad\qquad (2) \quad \alpha.P = \alpha.\tau.P$$

- The first would allow us to drop any $\tau$ actions of an agent
- The second would allow us to drop any but the first $\tau$.

Consider the following agents $A$ and $B$.

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




If (1) was valid, we should have $A = B$ but, looking at the transition graphs, they are different. So, we do not choose $P = \tau.P$ as an axiom.

The previous example indicated that (2) is intuitive, so (2) will be one of our axioms.

## 3. The Syntax of CCS

**We have**

- $\mathcal{A}$ – set of names
- $\overline{\mathcal{A}}$ – set of co-names
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ – set of labels, ranged over by $l$, $\bar{l}$.
- $Act = \mathcal{L} \cup \{\tau\}$ – set of actions, ranged over by $\alpha, \beta$.
- $\overline{\overline{a}} = a$, $\overline{\tau} = \tau$
- Relabelling function:

$$f(\bar{l}) = \overline{f(l)} \quad f(\tau) = \tau$$

**We further assume**

- $\mathcal{X}$ – set of agent variables, ranged over by $X, Y, \ldots$
- $\mathcal{K}$ – set of agent constants, ranged over by $A, B, \ldots,$
- We use $I$ or $J$ for indexing sets, for example $\{E_i : i \in I\}$ is a family of expressions indexed by $I$.

**Agent Expressions**

The set $\mathcal{E}$ of agent expressions is the smallest set which includes $\mathcal{X}$ and $\mathcal{K}$ and contains the following expressions, where $E$ and $E_i$ are already in $\mathcal{E}$:

1. $\alpha.E$, Prefix ($\alpha \in Act$)
2. $\sum_{i \in I} E_i$, Summation ($I$ is an indexing set)
3. $E_1 | E_2$, Parallel Composition
4. $E \backslash L$, Restriction ($L \subset \mathcal{L}$)
5. $E[f]$, Relabelling ($f$ is a relabelling function)

Summation is also known as Nondeterministic Choice.

**Rules for writing well formed agent expressions**

$\mathcal{E}$ is generated by the following rules:

1. If $X \in \mathcal{X}$, then $X \in \mathcal{E}$
2. If $A \in \mathcal{K}$, then $A \in \mathcal{E}$
3. If $E \in \mathcal{E}$ and $\alpha \in Act$, then $\alpha.E \in \mathcal{E}$
4. If $E_i \in \mathcal{E}$ for $i \in I$, the $\sum_{i \in I} E_i \in \mathcal{E}$
5. If $E_1, E_2 \in \mathcal{E}$, the $E_1|E_2 \in \mathcal{E}$
6. If $E \in \mathcal{E}$ and $L \subseteq \mathcal{L}$, then $E \backslash L \in \mathcal{E}$
7. If $E \in \mathcal{E}$ and $f$ is a relabelling function, then $E[f] \in \mathcal{E}$
8. $\mathcal{E}$ only contains those expressions constructed by using the above rules

We use $E$, $F$, $\ldots$ to range over $\mathcal{E}$.

**About summation**

- For any sets $I$ and $\{E_i \ : \ i \in I\}$, $\sum_{i \in I} E_i$ is the summation of all the $E_i$'s.
- When $I = \{1, 2\}$, $\sum_{i \in I} E_i$ is $E_1 + E_2$.
- When $I = \{1, 2, \ldots\}$, $\sum_{i \in I} E_i = E_1 + E_2 + \ldots$ .
- When $I = \emptyset$, $\mathbf{0} \stackrel{def}{=} \sum_{i \in I} E_i$ is the inactive agent.
- $\sum_{i \in I} E_i$ is also written as $\sum\{E_i \ : \ i \in I\}$.
- When $I$ is understood, $\tilde{E}$ for $\{E_i \ : \ i \in I\}$, and $\sum \tilde{E}$ or $\sum_i E_i$, for $\sum_{i \in I} E_i$.

**BNF definition of $\mathcal{E}$**

$\mathcal{E}$ is the set all $E$ defined by the following grammar

$$
\begin{array}{rcl}
E & ::= & A \\
 & \mid & X \\
 & \mid & \alpha.E \\
 & \mid & \sum_{i \in I} E \\
 & \mid & E|E \\
 & \mid & E\backslash L \\
 & \mid & E[f]
\end{array}
$$

Here $A \in \mathcal{K}$, $X \in \mathcal{X}$, $\alpha \in \mathcal{A}$, $L \subseteq \mathcal{L}$, and $f$ is a renaming function.

**Binding power of combinators**

The combinators have decreasing binding power in the order:

Restriction and Relabelling, Prefix, Parallel Composition and, finally, Summation

$$R + a.P|b.Q\backslash L \equiv R + ((a.P)|(b.(Q\backslash L)))$$

**Agent variables and agent constants**

- $Vars(E)$ denotes the set of (free) agent variables in $E$.
- An agent expression $E$ is called an agent if it does not contain agent variables.
- Each Constant is an agent, and has a defining equation of the form $A \stackrel{def}{=} P$. For example,

  $$A \stackrel{def}{=} a.A' \text{ and } A' \stackrel{def}{=} \overline{c}.A$$
- Agent constants can be defined in terms of each other, i.e. by mutual recursion.
- An agent expression which contains free agent variables represents (or can be thought of as) the set of agents obtained by different instantiations of these variables.

## 4. Operational Semantics of $\mathcal{E}$

We define the meaning (semantics) of an agent expression in terms of all its possible transitions.

We use the general notion of a labelled transition system (LTS) defined as follows.

A LTS is a triple $(S, T, \{\xrightarrow{t} :\ t \in T\})$ which consists of

- a set $S$ of states (or nodes),
- a set $T$ of (transition) labels, and
- a family of transition relations: $\xrightarrow{t} \subseteq S \times S$, for $t \in T$.

If $(s_1, s_2) \in \xrightarrow{t}$ for two states $s_1, s_2 \in S$ and a label $t \in T$, then

- we say that there is a transition from $s_1$ to $s_2$ by $t$, and
- denote this fact by $s_1 \xrightarrow{t} s_2$.

Thus, a LTS is a labelled directed graph

- the nodes of the graph are the states in $S$, and
- each edge is labelled with a label in $T$.

**The transition system for $\mathcal{E}$**

To define the semantics of $\mathcal{E}$ by a LTS, we

- take $S$ to be $\mathcal{E}$, the agent expressions,
- take $T$ to be $Act$, the actions,
- then define each transition relation $\xrightarrow{\alpha}$ over $\mathcal{E}$

These transition relations are defined by structural induction, i.e. by induction on the structure of agent expressions.

For example,

$$\text{From } A \xrightarrow{a} A' \text{ we infer } A|B \xrightarrow{a} A'|B$$

In general

$$\text{From } E \xrightarrow{\alpha} E' \text{ we infer } E|F \xrightarrow{\alpha} E'|F$$

We write this as

$$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \ \ldots \ \text{(one of the rules for } |)$$

Transition rules have the general form

$$\frac{\text{0 or more transitions called Hypotheses or Premises}}{\text{one transition called Conclusion}} \ \text{[Condition]}$$

- Transition rule for Prefix

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

- Transition rule for Summation

$$\textbf{Sum}_j \qquad \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j} \qquad (j \in I)$$

For $I = \{1, 2\}$

$$\textbf{Sum}_1 \qquad \frac{E_1 \xrightarrow{\alpha} E'_1}{E_1 + E_2 \xrightarrow{\alpha} E'_1}$$

$$\textbf{Sum}_2 \qquad \frac{E_2 \xrightarrow{\alpha} E'_2}{E_1 + E_2 \xrightarrow{\alpha} E'_2}$$

For $I = \emptyset$, no rule for $\mathbf{0} \stackrel{def}{=} \sum_{i \in I} E_i$.

This means that $\mathbf{0}$ does not have any transitions.

- Transition rules for Parallel Composition

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

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

$$\textbf{Com}_3 \qquad \frac{E \xrightarrow{l} E' \quad F \xrightarrow{\bar{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

- Transition rule for Restriction

$$\textbf{Res} \qquad \frac{E \stackrel{\alpha}{\rightarrow} E'}{E\backslash L \stackrel{\alpha}{\rightarrow} E'\backslash L} \quad (\alpha,\ \overline{\alpha} \notin L)$$

- Transition rule for Relabelling

$$\textbf{Rel} \qquad \frac{E \stackrel{\alpha}{\rightarrow} E'}{E[f] \stackrel{f(\alpha)}{\rightarrow} E'[f]}$$

- Transition Rule for Constants

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

For example, $A \stackrel{def}{=} a.A'$, so $A \stackrel{a}{\rightarrow} A'$.

**Summary of the rules**

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

**Sum$_j$** $$\frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i \in I} E_i \xrightarrow{\alpha} E'_j} \qquad (j \in I)$$

**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$_3$** $$\frac{E \xrightarrow{l} E' \quad F \xrightarrow{\overline{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

**Res** $$\frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L} \qquad (\alpha,\ \overline{\alpha} \notin L)$$

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

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

This set of rules is complete.

## Inference Trees

**Example**. Is the following a valid transition?

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

To answer this question, we shall try to construct an inference tree:

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

**Example**

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

where

$$A \stackrel{def}{=} a.A' \qquad B \stackrel{def}{=} c.B'$$

$$A' \stackrel{def}{=} \overline{c}.A \qquad B' \stackrel{def}{=} \overline{b}.B$$

$$\text{Res}\ \cfrac{\text{Com}_1\ \cfrac{\text{Con}\ \cfrac{\text{Act}\ \cfrac{}{a.A' \xrightarrow{a} A'}}{A \xrightarrow{a} A'}}{A|B \xrightarrow{a} A'|B}}{(A|B)\backslash c \xrightarrow{a} (A'|B)\backslash c}$$

**A closer look at inference trees**

Inference trees are the link between the operational semantics for agent constructs (operators) and transition graphs for agent expressions.

- Transition rules can be used to prove or disprove the correctness of transitions in a transition graph.
- We do this by constructing inference trees for these transitions using the transition rules.

Characteristics of inference trees:

- At the root of each successful tree will be the transition we are trying to prove.
- Each node consists of a transition labelled by the rule which was used to derive it.
- A node may only refer to transitions already proved correct.
- All leaf nodes in an inference tree must be instances of the rule **Act**, since this is the only rule with the empty set of hypotheses.
- If we try to build an inference tree for an invalid transition, then we will be unable to complete the tree according to the rules as just described.

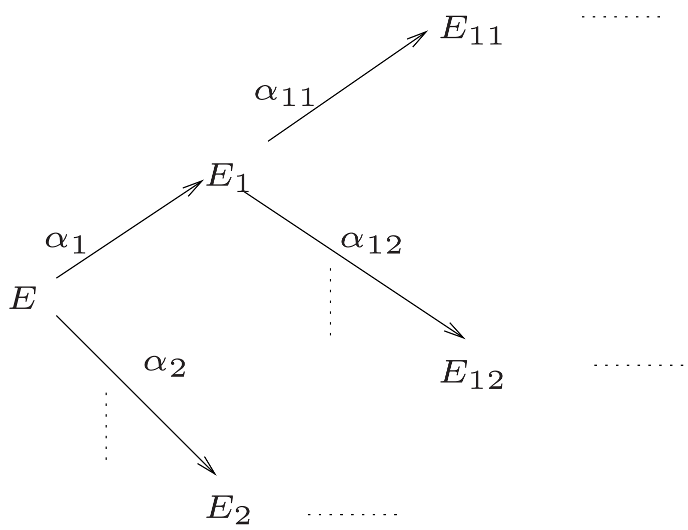
# 5. Derivatives & Derivation Trees

**Derivatives**

- Whenever $E \xrightarrow{\alpha} E'$, we call
  - the pair $(\alpha, E')$ an immediate derivative of $E$,
  - $\alpha$ an action of $E$, $E'$ an $\alpha$-derivative of $E$.
- Whenever $E \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} E'$, we call
  - $(\alpha_1 \ldots \alpha_n, E')$ a derivative of $E$,
  - $\alpha_1 \ldots \alpha_n$ an action-sequence of $E$, $E'$ an $\alpha_1 \ldots \alpha_n$-derivative of $E$.
- The empty sequence $\varepsilon$ is an action sequence of $E$, and $E$ itself is $\varepsilon$-derivative of $E$.

## Derivation trees

Collect the derivatives of $E$ into the derivation tree of $E$




- For each expression at a non-terminal node, its immediate derivations are represented by outgoing arcs.
- A tree may be finite or infinite.
- A tree is called total if the expressions at terminal nodes have no immediate derivatives.
- Otherwise, it is called partial.

**Example**

$$\begin{tikzcd}[column sep=large] & (c.E|F)\backslash a \\ ((a.c.E+b.\mathbf{0})|\overline{a}.F)\backslash a \arrow[ur, "\tau"] \arrow[dr, "b"'] & \\ & (\mathbf{0}|\overline{a}.F)\backslash a \end{tikzcd}$$

- This tree is partial.
- The lower terminal node $(\mathbf{0}|\overline{a}.F)\backslash a$ has no derivatives, whatever $F$ is.

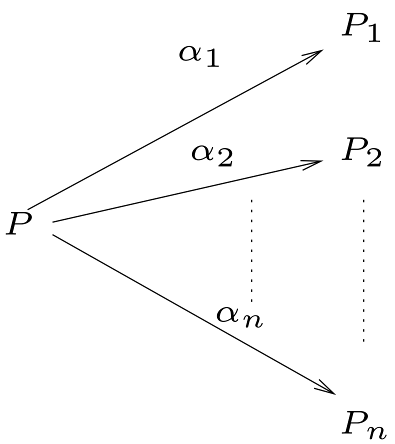
**Example**

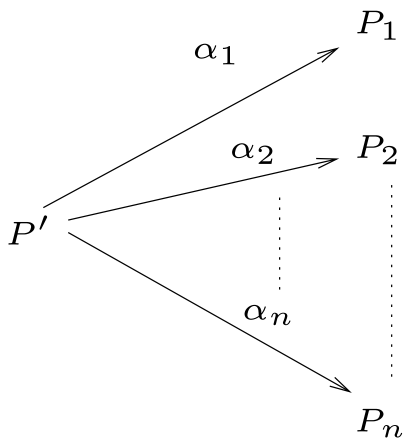
$$\begin{tikzcd}[column sep=large] & X \\ a.X+b.Y \arrow[ur, "a"] \arrow[dr, "b"] & \\ & Y \end{tikzcd}$$

- This tree is total but indeterminate.
- All immediate derivatives of agents are agents.

**Behavioural equivalence**

Consider agent expressions $P$ and $P'$

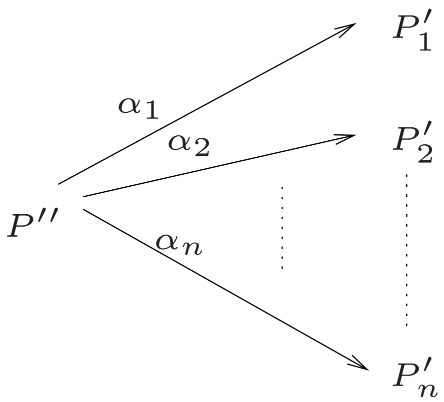



$$P \qquad\qquad P' \stackrel{def}{=} \sum\{\alpha_i.P_i : 1 \leq i \leq n\}$$

$P$ and $P'$ should be equivalent, though $P$ and $P'$ may be very different agent expressions.

More generally, if $P'_i$ is equivalent to $P_i$, and




then we should have $P$, $P'$ and $P''$ equivalent.

## 6. Sorts revisited

**Definition**:

For any $L \subseteq \mathcal{L}$, if the actions of $P$ and all its derivatives are members of $L \cup \{\tau\}$, then we say $P$ has sort $L$, and write $P : L$.

**Proposition 1**:

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 $E'$

**The syntactic sort of agent expressions**

Assign a sort to an expression

- Assign each $X$ the sort $\mathcal{L}(X)$
- Assign each Constant $A$ the sort $\mathcal{L}(A)$
- $\mathcal{L}(l.E) = \{l\} \cup \mathcal{L}(E)$
- $\mathcal{L}(\tau.E) = \mathcal{L}(E)$
- $\mathcal{L}(\sum_i E_i) = \bigcup_i \mathcal{L}(E_i)$
- $\mathcal{L}(E|F) = \mathcal{L}(E) \cup \mathcal{L}(F)$
- $\mathcal{L}(E\backslash L) = \mathcal{L}(E) - (L \cup \overline{L})$
- $\mathcal{L}(E[f]) = \{f(l) : l \in \mathcal{L}(E)\}$
- For $A \overset{def}{=} P$, $\mathcal{L}(P) \subseteq \mathcal{L}(A)$

**Questions and answers**

1. Is $\mathcal{L}(E)$ a sort of $E$?

   **Proposition 2**: Yes!

2. May $E$ perform all actions in $\mathcal{L}(E)$?

   **Not necessarily**!

   $\mathcal{L}(E)$ is called the syntactic sort of $E$.

3. Why do we need the notion of sort?

- Get an impression what actions $E$ may perform. For example, let $P$ be $((a.\mathbf{0} + b.\mathbf{0})|(\overline{b}.\mathbf{0} + c.\mathbf{0}))\backslash b$, then
$$\mathcal{L}(P) = \{a, c\}$$
- Some equational laws depend upon sorts, e.g.
$$(E|F)\backslash b = (E\backslash b)|F \quad \text{provided } b, \overline{b} \notin \mathcal{L}(F)$$

# 7. The Value-passing Calculus

**Example** : The buffer cell

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

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

Assume all values belong to a fixed set $V$

- $C'(x)$ becomes a family of Constants $\{C'_v : v \in V\}$.
- $\overline{out}(x)$. becomes a family of Prefixes $\{\overline{out_v}. : v \in V\}$.
- The single defining equation for $C'(x)$ becomes a family of defining equations

  $$\{C'_v \stackrel{def}{=} \overline{out_v}.C : \ v \in V\}$$
- '$in(x)$.' becomes '$\sum_{v \in V} in_v$.'
- Since '$in(x)$.' binds $x$ in $in(x).C'(x)$, the defining equation for $C$ becomes

  $$C \stackrel{def}{=} \sum_{v \in V} in_v.C'_v$$

The buffer cell can be defined as

$$C \stackrel{def}{=} \textstyle\sum_{v \in V} in_v.C'_v$$

$$C'_v \stackrel{def}{=} \overline{out_v}.C \qquad\qquad v \in V$$

**Example.** *Jobber*

$$J \stackrel{def}{=} in(j).St(j)$$

$$\begin{aligned} St(j) \stackrel{def}{=} \; & \textbf{if } e(j) \textbf{ then } F(j) \\ & \textbf{else if } h(j) \textbf{ then } UH(j) \\ & \textbf{else } UT(j) \end{aligned}$$

- The first defining equation becomes

$$J \stackrel{def}{=} \sum_{j \in V} in_j.St_j$$

- $St$ takes a parameter $j$. The second equation becomes a family of equations, one for each $j \in V$:

$$St_j \stackrel{def}{=} \begin{cases} F_j & (\text{if } easy(j)) \\ UH_j & (\text{if } \neg e(j) \wedge h(j)) \\ UT_j & (\text{if } \neg e(j) \wedge \neg h(j)) \end{cases}$$

- For each equation, the right-hand side is determined by the predicates *easy* and *hard*.

**Remarks**

- In theory, we do not need a larger calculus to deal with both value-passing and synchronisation.
- In practice, it is very tedious if systems with value-passing are always specified as families of defining equations.

**The full calculus**

- The intention is to enlarge the set $\mathcal{E}$ of agent expressions to a larger set $\mathcal{E}^+$ to express value-passing.
- We assign each agent Constant $A \in \mathcal{K}$ an arity, an non-negative integer representing the number of parameters which it takes.
- An $A \in \mathcal{K}$ with arity 0 does not carry any parameter.
- We assume value expressions $e$ and boolean expressions $b$, built from value variables $x, y, \ldots$ together with value constants $v$ by using value operators. E.G. $x + y$, $5 \times 2$, $(2 > 3) \wedge \neg(x \leq 3)$

**Agent expressions of the full calculus**

1. If $X \in \mathcal{X}$, then $X \in \mathcal{E}^+$
2. If $A \in \mathcal{K}$ with arity $n$, then $A(e_1, \ldots e_n) \in \mathcal{E}^+$
3. If $E \in \mathcal{E}^+$ and $a \in \mathcal{A}$, then $a(x).E, \overline{a}(e).E, \tau.E \in \mathcal{E}^+$
4. If $E_i \in \mathcal{E}^+$ for $i \in I$, then $\sum_{i \in I} E_i \in \mathcal{E}^+$
5. If $E_1, E_2 \in \mathcal{E}^+$, then $E_1 | E_2 \in \mathcal{E}^+$
6. If $E \in \mathcal{E}^+$ and $L \subseteq \mathcal{L}$, then $E \backslash L \in \mathcal{E}^+$
7. If $E \in \mathcal{E}^+$ and $f$ is a relabelling function, then $E[f] \in \mathcal{E}^+$
8. If $E \in \mathcal{E}^+$, then **if** $b$ **then** $E \in \mathcal{E}^+$
9. $\mathcal{E}^+$ only contains expressions constructed by using the above rules.
10. Each Constant $A$ has a defining equation

$$A(x_1, \ldots, x_n) \stackrel{def}{=} E$$

**Remark**

**if** $b$ **then** $E$ **else** $E'$ can be defined

$$\textbf{if } b \textbf{ then } E + \textbf{if } \neg b \textbf{ then } E'$$

**Syntactical equality**

For $E_1, E_2 \in \mathcal{E}^+$, we use $E_1 \equiv E_2$ to mean that $E_1$ and $E_2$ are syntactically identical.

Note the difference between '$\equiv$' and '$=$'

$$P + Q = Q + P \quad P|Q = Q|P \quad P + P = P$$

$$Jobshop = Strongjobber|Strongjobber$$

**But**

$$P + Q \not\equiv Q + P \quad P|Q \not\equiv Q|P \quad P + P \not\equiv P$$

$$Jobshop \not\equiv Strongjobber|Strongjobber$$

**Translation of $\mathcal{E}^+$ to $\mathcal{E}$**

- If $E$ has a free value variable $x$, then it can be treated as $\{E[v/x] : v \in V\}$.
- Thus, only deal with those $E$s which contain no free value variables.
- A value expression $e$ without variables is identical with the value $v$ to which it evaluates.

For $E \in \mathcal{E}^+$, let $\widehat{E}$ be the translated form in $\mathcal{E}$.

We define the translation recursively on the structure of agent expressions.

**The translation**

1. If $F \equiv X$, $\hat{F} \equiv X$
2. If $F \equiv a(x).E$, $\widehat{F} \equiv \sum_{v \in V} a_v.\widehat{E\{v/x\}}$
3. If $F \equiv \overline{a}(e).E$, $\widehat{F} \equiv \overline{a_e}.\widehat{E}$.
4. If $F \equiv \tau.E$, $\widehat{F} \equiv \tau.\widehat{E}$
5. If $F \equiv \sum_{i \in I} E_i$, $\widehat{F} \equiv \sum_{i \in I} \widehat{E_i}$
6. If $F \equiv E_1 | E_2$, $\widehat{F} \equiv \widehat{E_1} | \widehat{E_2}$
7. If $F \equiv E \backslash L$, $\widehat{F} \equiv \widehat{E} \backslash \{l_v : l \in L, v \in V\}$
8. If $F \equiv E[f]$, $\widehat{F} \equiv \widehat{E}[\hat{f}]$, where
   $$\hat{f}(l_v) = f(l)_v$$
9. If $F \equiv$ **if** $b$ **then** $E$, then
   $$\widehat{F} \equiv \begin{cases} \widehat{E} & \text{if } b = true \\ \mathbf{0} & \text{Otherwise} \end{cases}$$
10. If $F \equiv A(e_1, \ldots, e_n)$, $\widehat{F} \equiv A_{e_1, \ldots, e_n}$
11. A single defining equation $A(\tilde{x}) \stackrel{def}{=} E$ is translated into the indexed set of equations:
   $$\{A_{\tilde{v}} \stackrel{def}{=} \widehat{E\{\tilde{v}/\tilde{x}\}} : \tilde{v} \in V^n\}$$

**The semantics of $F$ is defined as that of $\widehat{F}$**

**Example**: The buffer cell in value-passing CCS

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

is translated into basic CCS as

$$C_\varepsilon \stackrel{def}{=} \sum_{v \in V} in_v.C'_v$$

$$C'_v \stackrel{def}{=} \overline{out_v}.C_\varepsilon \qquad \text{for all } v \in V$$

**Example**: The agent $B$

$$B \stackrel{def}{=} a(x).b(y).B(x,y)$$

$$B(x,y) \stackrel{def}{=} \overline{c}(x+1).\overline{d}(y+2).B$$

can also be defined in the basic CCS as

$$B_\varepsilon \stackrel{def}{=} \sum_{v_1 \in V} a_{v_1}.\sum_{v_2 \in V} b_{v_2}.B_{(v_1,v_2)}$$

$$B_{(v_1,v_2)} \stackrel{def}{=} \overline{c_{(v_1+1)}}.\overline{d_{(v_2+2)}}.B_\varepsilon \qquad \text{for all } v_1, v_2 \in V$$

## Summary of Chapter 2

- Syntax of the basic calculus
  - symbols for names – $\mathcal{A}$
  - symbols for labels – $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$
  - symbols for actions – $Act = \mathcal{L} \cup \{\tau\}$
  - The pre-emptive power of $\tau$
  - How to write valid agent expressions?
  - How can five combinators plus agent constants be used to write agent expressions for simple concurrent systems?

- Semantics of the basic calculus
  - The set of **transition rules** for CCS combinators, and for constants.
  - Transitions of agent expressions are inferred by constructing **inference trees**.
  - Inference trees are constructed using the transition rules.
  - The behaviour of systems is represented by transitions of their agent expressions, via an **LTS**.
  - Derivation trees and graphs and their use in verifying behavioural equivalence.
  - Formal definition of a **sort** for an agent expression, and the **syntactic sort** of an agent expression.
- Value-Passing
  - The syntax of the full calculus
  - Translation of the full calculus into the basic one
  - Semantics of the full calculus is defined in terms of semantics for the basic calculus.