17a - CCS syntax & op. semantics
CCS
Calculus of Communicating Systems
Sequential vs concurrent
Concurrency

IMP/HOFL (sequential paradigms)
- determinacy
- any two non-terminating programs are equivalent

concurrent paradigms
- exhibit intrinsic nondeterminism to external observers
- nontermination can be a desirable feature (e.g. servers)
- not all nonterminating processes are equivalent
- interaction is a primary issue
- new notions of behaviour / equivalence are needed
**CCS: basics**

Process algebra
- focus on few primitive operators (essential features)
- concise syntax to construct and compose processes
- not a full-fledged programming language
- full computational power (Turing equivalent)

Communication
- binary, message-passing over channels

Structural Operational Semantics
- small-step style (Labelled Transition System)
- processes as states
- ongoing interactions as labels
- defined by inference rules
- defined by induction on the structure of processes
Labelled transitions

ongoing interaction
with the environment
(with other processes)

\[ p \xrightarrow{\mu} q \]

a process
in its current state

the process
state after the
interaction

number of states/.transitions
can be infinite
Example: counter

\[ A_0 \quad \ldots \quad A_n \quad \text{val} \quad \rightarrow \quad A_{n+1} \quad \text{inc} \quad \rightarrow \quad \ldots \]

\[ \text{reset} \quad \downarrow \quad \text{stop} \]

\[ \text{Nil} \]
LTS: Labelled Transition System
**CCS: states and labels**

What is a process $\rho$?
- a sequential agent
- a system where many sequential agents interact

What is a label $\mu$?
- an action (e.g. an output)
- a dual action (e.g. an input)
- an internal action (silent action) (no interaction with the environment)

<table>
<thead>
<tr>
<th>send $v$ on channel $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha!v$</td>
</tr>
<tr>
<td>$\alpha?v$</td>
</tr>
<tr>
<td>receive $v$ on channel $\alpha$</td>
</tr>
<tr>
<td>$\tau$</td>
</tr>
</tbody>
</table>

concluded communication
**CCS: actions & coactions**

We can be even more abstract than that without losing computational expressiveness.

We disregard communicated values (imagine there is a dedicated channel for each value).

\[ \alpha!v \text{ becomes just } \overline{\alpha}_v \text{ or just } \overline{\alpha} \]

\[ \alpha?v \text{ becomes just } \alpha_v \text{ or just } \alpha \]

\[ \lambda \text{ denotes either } \alpha \text{ or } \overline{\alpha} \]

\[ \overline{\lambda} \text{ denotes its dual (assume } \overline{\overline{\alpha}} = \alpha \text{ )} \]
CCS: communication

\[
p_1 | p_2 \xrightarrow{\tau} q_1 | q_2
\]

\[
p_1 \xrightarrow{\lambda} q_1 \quad \text{and} \quad p_2 \xrightarrow{\bar{\lambda}} q_2
\]
Example: vending machine

- Student
  - drink

- HoldCup
  - coffee

- Select
  - coffee

- Tired
  - study
  - coin

- VendMach
  - coin
  - tea
  - cappuccino

- Serve_1
  - coin

- Serve_2
  - coin
CCS syntax
$p, q ::= \text{nil} \quad \text{inactive process}$

$\quad | x \quad \text{process variable (for recursion)}$

$\quad | \mu.p \quad \text{action prefix}$

$\quad | p\backslash\alpha \quad \text{restricted channel}$

$\quad | p[\phi] \quad \text{channel relabelling}$

$\quad | p + q \quad \text{nondeterministic choice (sum)}$

$\quad | p|q \quad \text{parallel composition}$

$\quad | \text{rec } x. \ p \quad \text{recursion}$

(operators are listed in order of precedence)
CCS: syntax

\[ p, q ::= \begin{array}{l}
\text{nil} \\
x \\
\mu.p \\
p\backslash\alpha \\
p[\phi] \\
p + q \\
p|q \\
\text{rec } x.\ p
\end{array} \]

\textbf{rec } x.\ \text{coffee}.x + \text{tea}.\text{nil} | \text{water}.\text{nil}

to be read as

\textbf{rec } x.\ (((\text{coffee}.x) + \text{tea}.\text{nil}) | \text{water}.\text{nil})

(operators are listed in order of precedence)
CCS: syntax

the only binder is the recursion operator

\[ \text{rec } x. \ p \]

the notion of free (process) variable is defined as usual

\[ \text{fv}(p) \]

a process is called \textit{closed} if it has no free variables

the notion of capture avoiding substitution is defined as usual

\[ p[q/x] \]

processes are taken up-to alpha-renaming of bound vars

\[ \text{rec } x. \ \text{coin}.x = \text{rec } y. \ \text{coin}.y \]
CCS operational semantics
**CCS: labels**

\( \mathcal{C} \) set of (input) actions, ranged by \( \alpha \)

\( \overline{\mathcal{C}} \) set of (output) co-actions, ranged by \( \overline{\alpha} \)

\( \Lambda = \mathcal{C} \cup \overline{\mathcal{C}} \) set of observable actions, ranged by \( \lambda \), \( \overline{\lambda} \)

\( \tau \not\in \Lambda \) a distinguished silent action

\( \mathcal{L} = \Lambda \cup \{\tau\} \) set of actions, ranged by \( \mu \)
LTS of a process

the LTS of CCS is infinite (one state for each process)

starting from $p$, consider all reachable states:
the LTS of a process can be finite/infinite
Nil process

\[ \text{nil} \not\rightarrow \]

the inactive process does nothing
no interaction is possible with the environment
represents a terminated agent
no operational semantics rule associated with \texttt{nil}
LTS of a process
Action prefix

\[
\begin{align*}
\text{Act) } & \quad \mu.p \xrightarrow{\mu} p \\
\end{align*}
\]

an action prefixed process can perform the action and continue as expected

the action may involve an interaction with the environment

\[
\overline{\text{coin.coffee.nil}}
\]

waits a coin, then gives a coffee and then it stops

\[
\begin{align*}
\overline{\text{coin.coffe.nil}} & \xrightarrow{\text{coin}} \overline{\text{coffe.nil}} & \overline{\text{coffe}} & \xrightarrow{} \text{nil} \\
\end{align*}
\]
LTS of a process

\[ \mu.p \xrightarrow{\mu} p \]
Nondeterministic choice

process \( p_1 + p_2 \) can behave either as \( p_1 \) or as \( p_2 \)

\[
\begin{align*}
\text{SumL)} \quad & \quad \frac{p_1 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q} \\
\text{SumR)} \quad & \quad \frac{p_2 \xrightarrow{\mu} q}{p_1 + p_2 \xrightarrow{\mu} q}
\end{align*}
\]

\[
\text{coin.}(\overline{\text{coffee}.\text{nil} + \text{tea}.\text{nil}})
\]

waits a coin, then gives a coffee or a tea, then it stops

\[
\begin{array}{c}
\text{coin.}(\overline{\text{coffee}.\text{nil} + \text{tea}.\text{nil}}) \\
\downarrow \text{coin} \\
\overline{\text{coffee}.\text{nil} + \text{tea}.\text{nil}} \\
\overline{\text{coffee}} \leftrightarrow \overline{\text{tea}} \\
\overline{\text{nil}}
\end{array}
\]
LTS of a process

$p + q$
Recursion

\[
\text{Rec)} \quad p[\text{rec } x. \ p / x] \xrightarrow{\mu} q \\
\text{rec } x. \ p \xrightarrow{\mu} q
\]

like a recursive definition \quad \text{let } x = p \text{ in } x

\[
\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})
\]
waits a coin, then gives a coffee and is ready again
or a tea and stops

\[
\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil}) \quad P \triangleq \text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})
\]

\[
\text{coffee.}(\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})) + \text{tea}.\text{nil}
\]

\[
\text{coffee}.P + \text{tea}.\text{nil}
\]

\[
\text{coffee.}(\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})) + \text{tea}.\text{nil}
\]

\[
\text{coffee}.P + \text{tea}.\text{nil}
\]

\[
\text{coffee}.(\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})) + \text{tea}.\text{nil}
\]

\[
\text{coffee}.P + \text{tea}.\text{nil}
\]

\[
\text{coffee}.(\text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil})) + \text{tea}.\text{nil}
\]

\[
\text{coffee}.P + \text{tea}.\text{nil}
\]
Recursion via process constants

Imagine some process constants $A$ are available together with a set $\Delta$ of declarations of the form

$$A \triangleq p$$

one for each constant

$$\begin{array}{c}
\text{Const)} \\
A \triangleq p \in \Delta \\
p \xrightarrow{\mu} q
\end{array}$$

$$A \xrightarrow{\mu} q$$

$$P \triangleq \text{coin.}(\overline{\text{coffee}}.P + \overline{\text{tea}}.\overline{\text{nil}})$$
CCS: capacity 1 buffer

\[ \Delta = \{ B_0^1 \triangleq \text{in}.B_1^1, B_1^1 \triangleq \text{out}.B_0^1 \} \]

\[ \text{rec } x. \text{in}.\text{out}.x \]
CCS: capacity 2 buffer

\[ B_0^2 \triangleq \text{in}.B_1^2 \]

\[ B_1^2 \triangleq \text{in}.B_2^2 + \text{out}.B_0^2 \]

\[ B_2^2 \triangleq \text{out}.B_1^2 \]
**CCS: boolean buffer**

\[ B_\emptyset \triangleq \text{in}_t.B_t + \text{in}_f.B_f \]

\[ B_t \triangleq \overline{\text{out}_t}.B_\emptyset \]

\[ B_f \triangleq \overline{\text{out}_f}.B_\emptyset \]
Parallel composition

processes running in parallel can interleave their actions or synchronize when dual actions are performed

\[ P \triangleq \text{coin} \cdot \text{coffee} \cdot \text{nil} \quad M \triangleq \text{coin} \cdot (\text{coffee} \cdot \text{nil} + \text{tea} \cdot \text{nil}) \]

\[ P \mid M \xrightarrow{\text{coin}} \text{coffee} \cdot \text{nil} \mid M \]

\[ P \mid M \xrightarrow{\text{coin}} P \mid (\text{coffee} \cdot \text{nil} + \text{tea} \cdot \text{nil}) \]

\[ P \mid M \xrightarrow{\tau} \text{coffee} \cdot \text{nil} \mid (\text{coffee} \cdot \text{nil} + \text{tea} \cdot \text{nil}) \]
LTS of a process

$p$

$q$

$p|q$
CCS: parallel buffers

\[ B_0^1 \triangleq \text{in.} B_1^1 \]

\[ B_1^1 \triangleq \text{out.} B_0^1 \]}
**CCS: parallel buffers**

\[
B^1_0 \triangleq \text{in}.B^1_1 \\
B^1_1 \triangleq \text{out}.B^1_0
\]
CCS: parallel buffers

\[ B_0^1 \triangleq \text{in}.B_1^1 \]

\[ B_1^1 \triangleq \text{out}.B_0^1 \]
CCS: parallel buffers

\[ B_0^1 \triangleq \text{in}.B_1^1 \]

\[ B_1^1 \triangleq \text{out}.B_0^1 \]
**CCS: parallel buffers**

$$B^1_0 \triangleq \text{in}.B^1_1$$

$$B^1_1 \triangleq \overline{\text{out}}.B^1_0$$

compare with the 2-capacity buffer
Restriction

\[
\begin{align*}
\text{Res) } & \quad p \xrightarrow{\mu} q \quad \mu \notin \{\alpha, \overline{\alpha}\} \\
& \quad p \backslash \alpha \xrightarrow{\mu} q \backslash \alpha
\end{align*}
\]

makes the channel \(\alpha\) private to \(p\)

no interaction on \(\alpha\) with the environment

if \(p\) is the parallel composition of processes, then they can synchronise on \(\alpha\)

\[
\begin{align*}
P & \triangleq \texttt{coin.coffee.nil} \\
M & \triangleq \texttt{coin.(coffee.nil + tea.nil)}
\end{align*}
\]

\[
\begin{align*}
(P|M) \backslash \texttt{coin \_ coffee \_ tea} & \xrightarrow{\tau} (\texttt{coffee.nil|coffee.nil + tea.nil}) \backslash \texttt{coin \_ coffee \_ tea} \\
(\texttt{coffee.nil|coffee.nil + tea.nil}) \backslash \texttt{coin \_ coffee \_ tea} & \xrightarrow{\tau} (\texttt{nil|nil}) \backslash \texttt{coin \_ coffee \_ tea}
\end{align*}
\]
Restriction: shorthand

given $S = \{\alpha_1, \ldots, \alpha_n\}$ we write $p\backslash S$

instead of $p\backslash \alpha_1 \ldots \backslash \alpha_n$

we omit trailing nil

$P \triangleq \text{coin.coffee}$ $\qquad M \triangleq \text{coin.}(\text{coffee} + \text{tea})$

$S \triangleq \{\text{coin, coffee, tea}\}$

$(P|M)\backslash S \xrightarrow{\tau} (\text{coffee}|\text{coffee} + \text{tea})\backslash S \xrightarrow{\tau} (\text{nil}|\text{nil})\backslash S$
LTS of a process
LTS of a process
Relabelling

\[
\begin{align*}
\text{Rel)} & \quad \frac{p \xrightarrow{\mu} q}{p[\phi] \xrightarrow{\phi(\mu)} q[\phi]} \\
\end{align*}
\]

renames the action channels according to \( \phi \)

we assume \( \phi(\tau) = \tau \) \quad \phi(\lambda) = \overline{\phi(\lambda)} \)

allows one to reuse processes

\[
P \triangleq \text{coin}.\text{coffee} \quad \phi(\text{coin}) = \text{moneta} \]
\[
\quad \phi(\text{coffee}) = \text{caffè} 
\]

\[
P[\phi] \xrightarrow{\text{moneta}} \text{coffee}[\phi] \xrightarrow{\text{caffè}} \text{nil}[\phi] 
\]
LTS of a process
LTS of a process

$p[\phi] \xrightarrow{\phi(\mu)}$
**CCS op. semantics**

### Act)
\[ \mu.p \xrightarrow{\mu} p \]

### Res)
\[ p \xrightarrow{\mu} q \quad \mu \not\in \{\alpha, \overline{\alpha}\} \]
\[ p \setminus \alpha \xrightarrow{\mu} q \setminus \alpha \]

### Rel)
\[ p \xrightarrow{\mu} q \]
\[ p[\phi] \xrightarrow{\phi(\mu)} q[\phi] \]

### SumL)
\[ p_1 \xrightarrow{\mu} q \]
\[ p_1 + p_2 \xrightarrow{\mu} q \]

### SumR)
\[ p_2 \xrightarrow{\mu} q \]
\[ p_1 + p_2 \xrightarrow{\mu} q \]

### ParL)
\[ p_1 \xrightarrow{\mu} q_1 \]
\[ p_1 \parallel p_2 \xrightarrow{\mu} q_1 \parallel p_2 \]

### Com)
\[ p_1 \xrightarrow{\lambda} q_1 \]
\[ p_2 \xrightarrow{\overline{\lambda}} q_2 \]
\[ p_1 \parallel p_2 \xrightarrow{\tau} q_1 \parallel q_2 \]

### ParR)
\[ p_2 \xrightarrow{\mu} q_2 \]
\[ p_1 \parallel p_2 \xrightarrow{\mu} p_1 \parallel q_2 \]

### Rec)
\[ p[\text{rec } x. \ p/x] \xrightarrow{\mu} q \]
\[ \text{rec } x. \ p \xrightarrow{\mu} q \]
Linked buffers

\[ B_0^1 \triangleq \text{in}.B_1^1 \quad \eta(\text{out}) = c \]
\[ B_1^1 \triangleq \overline{\text{out}}.B_0^1 \quad \phi(\text{in}) = c \]

\[
\begin{array}{ccc}
B_0^1[\phi] & \overset{\text{out}}{\longrightarrow} & \overline{c} \quad \overset{\text{in}}{\longrightarrow} \\
& \downarrow & \\
\overline{B_1^1[\phi]} & \overset{c}{\longrightarrow} & \overline{B_1^1[\eta]} \\
\end{array}
\]
Linked buffers

\[ B_0^1 \triangleq in.B_1^1 \quad \eta(out) = c \]

\[ B_1^1 \triangleq out.B_0^1 \quad \phi(in) = c \]
Linked buffers

\[ B_0^1 \triangleq in.B_1^1, \quad \eta(out) = c \]
\[ B_1^1 \triangleq out.B_0^1, \quad \phi(in) = c \]

\[ p \sim q \triangleq (p[\eta]|q[\phi]) \setminus c \]
Linked boolean buffers

\[ B_\emptyset \triangleq in_t.B_t + in_f.B_f \]
\[ B_t \triangleq \overline{out_t}.B_\emptyset \]
\[ B_f \triangleq \overline{out_f}.B_\emptyset \]
\[ \eta(out_t) = c_t \quad \phi(in_t) = c_t \]
\[ \eta(out_f) = c_f \quad \phi(in_f) = c_f \]
\[ p \sim q \triangleq (p[\eta]|q[\phi])\setminus\{c_t, c_f\} \]
Linked boolean buffers

\[ B_\emptyset \triangleq \text{in}_t B_t + \text{in}_f B_f \]

\[ B_t \triangleq \overline{\text{out}_t B_\emptyset} \]

\[ B_f \triangleq \overline{\text{out}_f B_\emptyset} \]
Linked boolean buffers

\[ B_\emptyset \triangleq in_t.B_t + in_f.B_f \]
\[ \eta(out_t) = c_t \quad \phi(in_t) = c_t \]
\[ B_t \triangleq \overline{out_t}.B_\emptyset \]
\[ \eta(out_f) = c_f \quad \phi(in_f) = c_f \]
\[ B_f \triangleq \overline{out_f}.B_\emptyset \]
\[ p \sim q \triangleq (p[\eta]|q[\phi])\setminus\{c_t, c_f\} \]
Linked boolean buffers

\[ B_\emptyset \triangleq \in_t. B_t + \in_f. B_f \]

\[ \eta(out_t) = c_t \quad \phi(in_t) = c_t \]

\[ B_t \triangleq \overline{out_t}. B_\emptyset \]

\[ \eta(out_f) = c_f \quad \phi(in_f) = c_f \]

\[ B_f \triangleq \overline{out_f}. B_\emptyset \]

\[ p \sim q \triangleq (p[\eta]|q[\phi]) \backslash \{c_t, c_f\} \]
 CCS with value passing

\[
\alpha!v.p \xrightarrow{\alpha_v} p
\]

\[
\alpha?x.p \xrightarrow{\alpha_v} p[v/x]
\]

when the set of values is finite \( V \triangleq \{v_1, \ldots, v_n\} \)

\[
\alpha!v.p \equiv \overline{\alpha_v}.p
\]

\[
\alpha?x.p \equiv \alpha_{v_1}.p[v_1/x] + \cdots + \alpha_{v_n}.p[v_n/x]
\]

receive

\[
\begin{align*}
    v & \rightarrow p \\
    w & \rightarrow q \\
    - & \rightarrow r
\end{align*}
\]

\[
\equiv \alpha_v.p + \alpha_w.q + \sum_{z \neq v, w} \alpha_z.r
\]

end
Exercise: LTS?

\[ P \triangleq (\text{rec } x. \alpha.x) + (\text{rec } x. \beta.x) \]
Exercise: LTS?

\[ Q \triangleq \text{rec } x. (\alpha.x + \beta.x) \]

\[ Q \triangleq \text{rec } x. \alpha.x + \beta.x \]

\[ Q \triangleq \alpha.Q + \beta.Q \]

\[ \alpha \bowtie Q \bowtie \beta \]
Exercise: LTS?

\[ R \triangleq \text{rec } x. (\alpha.x + \beta.\text{nil}) \]
\[ R \triangleq \text{rec } x. \alpha.x + \beta \]
\[ R \triangleq \alpha.R + \beta \]

\[ \alpha \xrightarrow{} R \]
\[ \beta \]
\[ \text{nil} \]
Exercise: LTS?

\[ T \triangleq \text{rec } x. \ (\alpha \cdot \text{nil}|x) + \beta \cdot \text{nil} \]

\[ T \triangleq \text{rec } x. \ (\alpha|x) + \beta \]

\[ T \triangleq (\alpha|T) + \beta \]

\[ \begin{array}{ccc}
\text{nil} & \alpha|\text{nil} & \alpha|\alpha|\text{nil} \\
\beta & \beta & \beta \\
\alpha & \alpha & \alpha \\
\text{nil}|T & \alpha|\text{nil}|T & \alpha|\alpha|\text{nil}|T \\
\end{array} \]

\[ \begin{array}{ccc}
\text{nil} & \alpha|\text{nil} & \alpha|\alpha|\text{nil} \\
\beta & \beta & \beta \\
\alpha & \alpha & \alpha \\
\text{nil}|T & \alpha|\text{nil}|T & \alpha|\alpha|\text{nil}|T \\
\end{array} \]
Exercise: LTS?

\[ U \triangleq \text{rec } x. \ ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \ \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]

\[ U \xrightarrow{\beta} \alpha|U \]

\[ \alpha \]

\[ \text{nil}|\beta.U \]
Exercise: LTS?

\[ U \triangleq \text{rec } x. \ (\alpha . \text{nil} ) \mid \beta . x \]

\[ U \triangleq \text{rec } x. \ \alpha \mid \beta . x \]

\[ U \triangleq \alpha \mid \beta . U \]

\[ \begin{array}{c}
  U \\
  \downarrow \alpha \\
  \text{nil} \mid \beta . U \\
  \downarrow \beta \\
  \text{nil} \mid U
\end{array} \]

\[ \beta \]

\[ \rightarrow \alpha \mid U \]
Exercise: LTS?

\[ U \triangleq \text{rec } x. ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]

\[
\begin{align*}
U & \xrightarrow{\beta} \alpha|U \xrightarrow{\beta} \alpha|\alpha|U \\
\end{align*}
\]

\[
\begin{align*}
\alpha & \quad \alpha \\
\text{nil}|\beta.U & \xrightarrow{\alpha} \alpha|\text{nil}|\beta.U \\
\beta & \quad \beta \\
\text{nil}|U & \xrightarrow{\beta} \text{nil}|U
\end{align*}
\]
Exercise: LTS?

\[ U \triangleq \text{rec } x. ((\alpha \cdot \text{nil}) \mid \beta . x) \]

\[ U \triangleq \text{rec } x. \alpha \mid \beta . x \]

\[ U \triangleq \alpha \mid \beta . U \]

\[ U \xrightarrow{\beta} \alpha \mid U \xrightarrow{\beta} \alpha \mid \alpha \mid U \]

\[ \alpha \]

\[ \beta \]

\[ \text{nil} \mid \beta . U \xrightarrow{\alpha} \alpha \mid \text{nil} \mid \beta . U \xrightarrow{\beta} \alpha \mid \text{nil} \mid U \]

\[ \beta \]

\[ \alpha \]

\[ \beta \]

\[ \alpha \]

\[ \text{nil} \mid U \]

\[ \text{nil} \mid \text{nil} \mid \beta . U \]
Exercise: LTS?

\[ U \triangleq \text{rec } x. \ ( (\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \ \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]

\[ U \quad \xrightarrow{\beta} \quad \alpha|U \quad \xrightarrow{\beta} \quad \alpha|\alpha|U \quad \xrightarrow{\beta} \quad \ldots \]

\[ \quad \xrightarrow{\alpha} \quad \text{nil}|\beta.U \quad \xrightarrow{\alpha} \quad \alpha|\text{nil}|\beta.U \quad \xrightarrow{\beta} \quad \alpha|\text{nil}|U \quad \xrightarrow{\ldots} \]

\[ \quad \xrightarrow{\beta} \quad \text{nil}|U \quad \xrightarrow{\alpha} \quad \text{nil}|\text{nil}|\beta.U \]
Exercise: LTS?

\[ U \triangleq \text{rec } x. \ ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \ \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]

\[ U \xrightarrow{\beta} \alpha|U \xrightarrow{\beta} \alpha|\alpha|U \xrightarrow{\beta} \ldots \]

\[ \text{nil}|\beta.U \xrightarrow{\alpha} \alpha|\text{nil}|\beta.U \xrightarrow{\beta} \alpha|\text{nil}|U \xrightarrow{\beta} \ldots \]

\[ \text{nil}|U \xrightarrow{\beta} \text{nil}|\beta.U \xrightarrow{\alpha} \alpha|\text{nil}|\beta.U \xrightarrow{\beta} \alpha|\text{nil}|U \xrightarrow{\beta} \ldots \]

\[ \text{nil}|\text{nil}|U \xrightarrow{\alpha} \alpha|\text{nil}|U \xrightarrow{\beta} \ldots \]
Exercise: LTS?

let's ignore nil

\[ U \triangleq \text{rec } x. \ ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \ \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]
Exercise: LTS?

let's ignore nil

\[ U \triangleq \text{rec } x. ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]

\[ \begin{array}{c}
U \\
\xrightarrow{\beta} \\
\alpha|U \\
\end{array} \]

\[ \begin{array}{c}
\alpha \\
\xleftarrow{} \\
\beta \\
\xleftarrow{} \\
\beta.U \\
\end{array} \]
Exercise: LTS?

let’s ignore nil

\[ U \triangleq \text{rec } x. \ (\alpha . \text{nil}) \mid \beta . x \]

\[ U \triangleq \text{rec } x. \ \alpha \mid \beta . x \]

\[ U \triangleq \alpha \mid \beta . U \]
Exercise: LTS?

let's ignore $\textit{nil}$

$$U \triangleq \text{rec } x. \ ((\alpha.\textit{nil})|\beta.x)$$

$$U \triangleq \text{rec } x. \ \alpha|\beta.x$$

$$U \triangleq \alpha|\beta.U$$

\[
\begin{array}{c}
\xymatrix{
U \ar[r]^-{\beta} & \alpha|U \\
\alpha(U) \ar[r]^-{\alpha} & \beta(U) \\
\beta(U) \ar[r]_-{\alpha} & \alpha|\beta.U \\
\alpha(U) \ar[r]_-{\beta} & \beta(U)
}
\end{array}
\]
Exercise: LTS?

let's ignore \textbf{nil}

\[ U \triangleq \text{rec } x. \ ((\alpha.\text{nil})|\beta.x) \]

\[ U \triangleq \text{rec } x. \ \alpha|\beta.x \]

\[ U \triangleq \alpha|\beta.U \]
Write an interactive counter modulo 4 in CCS

The counter process has four input channels: \textit{inc, val, reset, stop}

and four output channels:
\[ c_0, c_1, c_2, c_3 \]

used to display the current value of the counter

Draw the LTS of the counter process.