PSC 2023/24 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

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
From your forms

(over 8 answers)
Labelled transitions

ongoing interaction with the environment (with other processes)

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

a process in its current state

number of states/transitions can be infinite

the process state after the interaction
Example: counter

\[ A_0 \quad \cdots \quad A_n \quad \cdots \]

\[ \xrightarrow{\text{val}} \]

\[ A_n \quad \xrightarrow{\text{inc}} \quad A_{n+1} \quad \cdots \]

\[ \xrightarrow{\text{reset}} \quad \text{Nil} \]

\[ \xrightarrow{\text{stop}} \]
LTS: Labelled Transition System
CCS: states and labels

What is a process $p$?
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)

send $v$ on channel $\alpha$

$\alpha!v$

receive $v$ on channel $\alpha$

$\alpha?v$

$\tau$

concluded communication

10
**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).

\(\alpha!v\) becomes just \(\overline{\alpha_v}\) or just \(\overline{\alpha}\)

\(\alpha?v\) becomes just \(\alpha_v\) or just \(\alpha\)

\(\lambda\) denotes either \(\alpha\) \(\overline{\alpha}\)

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

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

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

- Tired
  - study
  - coin
- Student
  - drink
- HoldCup
  - coffee
- Select
  - coffee

VendMach
- coin
- tea
- cappuccino

Serve_1
- coin

Serve_2
CCS syntax
From your forms

(over 8 answers)
CCS: syntax

\[ p, q ::= \text{nil} \quad \text{inactive process} \]
\[ x \quad \text{process variable (for recursion)} \]
\[ \mu.p \quad \text{action prefix} \]
\[ p\backslash \alpha \quad \text{restricted channel} \]
\[ p[\phi] \quad \text{channel relabelling} \]
\[ p + q \quad \text{nondeterministic choice (sum)} \]
\[ p|q \quad \text{parallel composition} \]
\[ \text{rec } x. \ p \quad \text{recursion} \]

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

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

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

to be read as

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

(operators are listed in order of precedence)
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

\[ \mathcal{C} \quad \text{set of (input) actions, ranged by } \alpha \]
\[ \overline{\mathcal{C}} \quad \text{set of (output) co-actions, ranged by } \overline{\alpha} \]
\[ \Lambda = \mathcal{C} \cup \overline{\mathcal{C}} \quad \text{set of observable actions, ranged by } \lambda \overline{\lambda} \]
\[ \tau \notin \Lambda \quad \text{a distinguished silent action} \]
\[ \mathcal{L} = \Lambda \cup \{\tau\} \quad \text{set of actions, ranged by } \mu \]
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 \text{nil}
LTS of a process

nil
Action prefix

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

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

the action may involve an interaction with the environment

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

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

\[\text{coin.coffee.nil} \xrightarrow{\text{coin}} \text{coffee.nil} \xrightarrow{\text{coffee}} \text{nil}\]
LTS of a process

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

process $p_1 + p_2$ can behave either as $p_1$ or as $p_2$

$\text{coin.} (\text{coffee.nil + tea.nil})$

waits a coin, then gives a coffee or a tea, then it stops
LTS of a process

\[ p + q \]
Recursion

\[
\begin{array}{c}
\text{Rec) } p \left[ \text{rec } x. \ p / x \right] \xrightarrow{\mu} q \\
\text{rec } x. \ p \xrightarrow{\mu} q
\end{array}
\]

like a recursive definition  
let \( x = p \) 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

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

\[
\begin{array}{c}
P \triangleq \text{rec } x. \ \text{coin.}(\text{coffee}.x + \text{tea}.\text{nil}) \\
\text{coffee.}P + \text{tea}.\text{nil}
\end{array}
\]
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

$$\begin{align*}
\text{Const) } & \quad A \triangleq p \in \Delta \quad p \xrightarrow{\mu} q \\
& \quad \frac{}{A \xrightarrow{\mu} q}
\end{align*}$$

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

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

\text{rec } x. \text{in.out}.x

\[
\begin{array}{c}
\begin{array}{c}
\text{in} \\
B_0^1 \\
\text{out}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
B_0^1 \\
\text{out} \\
\text{in} \\
B_1^1
\end{array}
\end{array}
\]
**CCS: capacity 2 buffer**

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

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

\[ P|M \xrightarrow{\text{coin}} \text{coffee.nil|M} \]

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

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

$p\parallel q$
**CCS: parallel buffers**

\[ B_0^1 \triangleq in.B_1^1 \]

\[ B_1^1 \triangleq \overline{out}.B_0^1 \]

\[ B_0^1 \mid B_0^1 \]
CCS: parallel buffers

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

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

\[ B_1^1|B_0^1 \]

\[ B_0^1|B_0^1 \]

\[ B_0^1|B_1^1 \]
CCS: parallel buffers

\[
\begin{align*}
\mathcal{B}_0^1 & \triangleq in.\mathcal{B}_1^1 \\
\mathcal{B}_1^1 & \triangleq out.\mathcal{B}_0^1
\end{align*}
\]
CCS: parallel buffers

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

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

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

compare with the 2-capacity buffer
Restriction

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

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 \( \alpha \)

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

\[
(P|M) \setminus \text{coin} \setminus \text{coffee} \setminus \text{tea} \xrightarrow{\tau} (\text{coffee.nil}|\text{coffee.nil + tea.nil}) \setminus \text{coin} \setminus \text{coffee} \setminus \text{tea}
\]

\[
(\text{coffee.nil}|\text{coffee.nil + tea.nil}) \setminus \text{coin} \setminus \text{coffee} \setminus \text{tea} \xrightarrow{\tau} (\text{nil}|\text{nil}) \setminus \text{coin} \setminus \text{coffee} \setminus \text{tea}
\]
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 \( \text{nil} \)

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

\[
S \triangleq \{\text{coin, coffee, tea}\}
\]

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

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

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

we assume

\[
\phi(\tau) = \tau \quad \phi(\lambda) = \overline{\phi(\lambda)}
\]

allows one to reuse processes

\[
P \triangleq \text{coin.coffee}
\]

\[
\phi(\text{coin}) = \text{moneta}
\]

\[
\phi(\text{coffee}) = \text{caffè}
\]

\[
P[\phi] \xrightarrow{\text{moneta}} \text{coffee}[\phi] \xrightarrow{\text{caffè}} \text{nil}[\phi]
\]
LTS of a process
LTS of a process
**CCS op. semantics**

- **Act)** $\mu.p \xrightarrow{\mu} p$
- **Res)** $p \xrightarrow{\mu} q \quad \mu \not\in \{\alpha, \overline{\alpha}\}$
- **Rel)** $p \xrightarrow{\mu} q$
- **SumL)** $p_1 \xrightarrow{\mu} q$ \quad $p_1 + p_2 \xrightarrow{\mu} q$
- **SumR)** $p_2 \xrightarrow{\mu} q$
- **ParL)** $p_1 \xrightarrow{\mu} q_1$ \quad $p_1 \parallel p_2 \xrightarrow{\mu} q_1 \parallel p_2$
- **Com)** $p_1 \xrightarrow{\lambda} q_1 \quad p_2 \xrightarrow{\overline{\lambda}} q_2$
- **ParR)** $p_2 \xrightarrow{\mu} q_2$
- **Rec)** $p[\text{rec } x. p/x] \xrightarrow{\mu} q$
- $\text{rec } x. p \xrightarrow{\mu} q$

$\text{parL)}$ $\quad \text{ParR)}$ $\quad \text{Com)}$ $\quad \text{Rec)}$
Linked buffers

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

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

\[ B_0^1[\phi] \]

\[ B_0^1[\eta] \]

\[ B_1^1[\phi] \]

\[ B_1^1[\eta] \]
Linked buffers

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

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

\[ B^1_0 \triangleq \text{in}.B^1_1 \quad \eta(\text{out}) = c \]

\[ B^1_1 \triangleq \overline{\text{out}}.B^1_0 \quad \phi(\text{in}) = c \]

\[ p \prec q \triangleq (p[\eta]\mid q[\phi]) \backslash c \]
Linked boolean buffers

\[ B_0 \triangleq in_t.B_t + in_f.B_f \quad \eta(out_t) = c_t \quad \phi(in_t) = c_t \]

\[ B_t \triangleq \overline{out_t}.B_0 \quad \eta(out_f) = c_f \quad \phi(in_f) = c_f \]

\[ B_f \triangleq \overline{out_f}.B_0 \quad p \sim q \triangleq (p[\eta]q[\phi]) \backslash \{c_t, c_f\} \]
Linked boolean buffers

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

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

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

\[ B_\emptyset[\eta] \]

\[ B_\emptyset[\phi] \]
Linked boolean buffers

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

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

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

\[ \eta(\text{out}_t) = c_t \quad \phi(\text{in}_t) = c_t \]

\[ \eta(\text{out}_f) = c_f \quad \phi(\text{in}_f) = c_f \]

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

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

end
Exercise: LTS?

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

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

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

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

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

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

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

\[ R \triangleq \alpha.R + \beta \]
Exercise: LTS?

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

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

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

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

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

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

[Diagram]

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

\[
\alpha
\]

\[
\text{nil}|\beta.U
\]

\[
\beta
\]

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

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

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

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

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

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

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

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

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

\[ \alpha \]

\[ \beta \]

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

\[ \beta \]

\[ \alpha \]

\[ \text{nil}|U \]

\[ \alpha \]

\[ \beta \]

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

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

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

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

\[
\begin{array}{ccc}
U & \xrightarrow{\beta} & \alpha \cdot U \\
& \downarrow{\alpha} & \downarrow{\alpha} \\
\text{nil} | \beta \cdot U & \xrightarrow{\alpha} & \alpha \cdot \text{nil} | \beta \cdot U \\
& \downarrow{\beta} & \downarrow{\alpha} \\
\text{nil} | U & \xrightarrow{\alpha} & \text{nil} | \text{nil} | \beta \cdot U
\end{array}
\]

\[
\begin{array}{ccc}
U & \xrightarrow{\beta} & \alpha \cdot U \\
& \downarrow{\alpha} & \downarrow{\alpha} \\
\alpha \cdot U & \xrightarrow{\alpha} & \alpha \cdot \alpha \cdot U \\
& \downarrow{\alpha} & \downarrow{\alpha} \\
\alpha \cdot \alpha \cdot U & \xrightarrow{\alpha} & \alpha \cdot \alpha \cdot \alpha \cdot U \\
& \downarrow{\alpha} & \downarrow{\alpha} \\
\cdots & \cdots & \cdots
\end{array}
\]
Exercise: LTS?

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

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

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

\[ \begin{array}{c}
  U \xrightarrow{\beta} \alpha|U \xrightarrow{\beta} \alpha|\alpha|U \xrightarrow{\beta} \cdots \\
  \alpha \downarrow \quad \alpha \downarrow \quad \alpha \downarrow \\
  \alpha|\beta.U \xrightarrow{\alpha} \alpha|\text{nil}|\beta.U \xrightarrow{\beta} \alpha|\text{nil}|U \xrightarrow{\beta} \cdots \\
  \beta \downarrow \quad \alpha \downarrow \quad \beta \downarrow \\
  \text{nil}|U \xrightarrow{\alpha} \text{nil}|\beta.U \xrightarrow{\alpha} \text{nil}|\text{nil}|\beta.U \xrightarrow{\alpha} \cdots \\
  \beta \downarrow \quad \alpha \downarrow \quad \beta \downarrow \\
  \text{nil}|\text{nil}|U \xrightarrow{\alpha} \text{nil}|\text{nil}|U \xrightarrow{\alpha} \cdots 
\end{array} \]
Exercise: LTS?

let’s ignore \texttt{nil}

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

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

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

let’s ignore \texttt{nil}

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

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

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

let's ignore nil

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

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

\[ U \triangleq \alpha|\beta.U \]
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: \texttt{LTS}?

let's ignore \texttt{nil}

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

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

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

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.