PSC 2022/23 (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 15 answers)
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

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

\[ \overset{\text{inc}}{\rightarrow} \]

\[ \overset{\text{reset}}{\rightarrow} \]

\[ \text{Nil} \]

\[ \overset{\text{stop}}{\rightarrow} \]
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 $\alpha$

$\alpha?v$

concluded communication

$\tau$
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 \( \overline{\overline{\alpha}} = \alpha \) )
CCS: communication

$p_1 | p_2 \xrightarrow{\tau} q_1 | q_2$

$p_1 \xrightarrow{\lambda} q_1$

$p_2 \xrightarrow{\bar{\lambda}} q_2$
Example: vending machine

Student ← [drink] HoldCup ← [coffee] Select

Tired

[study] coin

VendMach

coint

cappuccino

tea

Serve₁

Serve₂
CCS syntax
From your forms

(over 15 answers)
CCS: syntax

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

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

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

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

to be read as

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

\[ \tau \notin \Lambda \] 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 \text{nil}
LTS of a process

nil
Action prefix

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

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

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

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

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

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

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

\[
\text{coin} \downarrow
\]

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

\[
\text{coffee} \quad \text{tea} \\
\quad \downarrow \quad \downarrow
\]

\[
\text{nil}
\]
LTS of a process

\[ p \]

\[ q \]

\[ p + q \]
Recursion

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

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

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

waits a coin, then gives a coffee and is ready again
or a tea and stops

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

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

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

\[
\text{tea}
\]

\[
\text{nil}
\]

\[
\text{tea}
\]

\[
\text{nil}
\]

28
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

$$\text{Const)} \quad \frac{A \triangleq p \in \Delta \quad p \xrightarrow{\mu} q}{A \xrightarrow{\mu} q}$$

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

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

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

\[ B_0^1 \]

\[ B_1^1 \]

\[ \text{in} \]

\[ \text{out} \]
**CCS: capacity 2 buffer**

\[
\begin{align*}
B_0^2 &\triangleq in.B_1^2 \\
B_1^2 &\triangleq in.B_2^2 + out.B_0^2 \\
B_2^2 &\triangleq out.B_1^2
\end{align*}
\]
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

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

P\mid M & \xrightarrow{\text{coin}} \text{coffee.nil}\mid M \\

P\mid M & \xrightarrow{\text{coin}} P\mid (\text{coffee.nil} + \text{tea.nil}) \\

P\mid M & \xrightarrow{\tau} \text{coffee.nil}\mid(\text{coffee.nil} + \text{tea.nil})
\end{align*}
\]
LTS of a process
CCS: parallel buffers

\[ B_0^1 \triangleq in.B_1^1 \]
\[ B_1^1 \triangleq out.B_0^1 \]
CCS: parallel buffers

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

\[ B^1_1 \triangleq \text{out}.B^1_0 \]
$B_0^1 \triangleq in.B_1^1$

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

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

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

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

\[ B_0^2 \]

\[ B_1^2 \]

\[ B_2^2 \]

\[ B_0^1 \mid B_0^1 \]

\[ B_1^1 \mid B_1^1 \]

\[ B_0^1 \mid B_1^1 \]

\[ B_1^1 \mid B_0^1 \]

compare with the 2-capacity buffer
Restriction

\[ p \xrightarrow{\mu} q \quad \mu \not\in \{\alpha, \overline{\alpha}\} \]

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

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 \( \alpha \)

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

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

\[ (\text{coffee.nil|coffee.nil + tea.nil})\backslash \text{coin}\backslash \text{coffee}\backslash \text{tea} \xrightarrow{\tau} (\text{nil|nil})\backslash \text{coin}\backslash \text{coffee}\backslash \text{tea} \]
Restriction: shorthand

given \( S = \{\alpha_1, \ldots, \alpha_n\} \) we write \( p \setminus S \)

instead of \( p \setminus \alpha_1 \ldots \setminus \alpha_n \)

we omit trailing \( \text{nil} \)

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

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

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

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

we assume \( \phi(\tau) = \tau \) \hspace{1cm} \( \phi(\overline{\lambda}) = \overline{\phi(\lambda)} \)

allows one to reuse processes

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

\[
\begin{align*}
\phi(\text{coin}) &= \text{moneta} \\
\phi(\text{coffee}) &= \text{caffè}
\end{align*}
\]

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

\[ p \xrightarrow{\mu} \]

\[ \]
LTS of a process
CCS op. semantics

\[
\begin{align*}
\text{Act)} & \quad \mu.p \xrightarrow{\mu} p \\
\text{Res)} & \quad p \xrightarrow{\mu} q \quad \mu \not\in \{\alpha, \bar{\alpha}\} \\
& \quad p \setminus \alpha \xrightarrow{\mu} q \setminus \alpha \\
\text{Rel)} & \quad p \xrightarrow{\mu} q \\
& \quad p[\phi] \xrightarrow{\phi(\mu)} q[\phi] \\
\text{SumL)} & \quad p_1 \xrightarrow{\mu} q \\
& \quad p_1 + p_2 \xrightarrow{\mu} q \\
\text{SumR)} & \quad p_2 \xrightarrow{\mu} q \\
& \quad p_1 + p_2 \xrightarrow{\mu} q \\
\text{ParL)} & \quad p_1 \xrightarrow{\mu} q_1 \\
& \quad p_1 | p_2 \xrightarrow{\mu} q_1 | p_2 \\
\text{Com)} & \quad p_1 \xrightarrow{\lambda} q_1 \\
& \quad p_2 \xrightarrow{\bar{\lambda}} q_2 \\
& \quad p_1 | p_2 \xrightarrow{\tau} q_1 | q_2 \\
\text{ParR)} & \quad p_2 \xrightarrow{\mu} q_2 \\
& \quad p_1 | p_2 \xrightarrow{\mu} p_1 | q_2 \\
\text{Rec)} & \quad p[\text{rec } x.\ p/x] \xrightarrow{\mu} q \\
& \quad \text{rec } x.\ p \xrightarrow{\mu} q
\end{align*}
\]
Linked buffers

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

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

\[ (B_0^1[\eta]|B_0^1[\phi]) \backslash \text{c} \]

\[ (B_1^1[\eta]|B_1^1[\phi]) \backslash \text{c} \]

\[ \tau \]

\[ (B_0^1[\eta]|B_1^1[\phi]) \backslash \text{c} \]

\[ (B_1^1[\eta]|B_0^1[\phi]) \backslash \text{c} \]
Linked buffers

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

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

\[
\begin{array}{ccc}
\quad & B_0^1[\phi] & \quad \\
\quad & \text{out} & \quad \\
\iddots & \iddots & \iddots \\
\quad & B_1^1[\eta] & \quad \\
\quad & \text{in} & \quad \\
\end{array}
\]
Linked buffers

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

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

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

\[ B_{\emptyset} \triangleq in_t.B_t + in_f.B_f \quad \eta(out_t) = c_t \quad \phi(in_t) = c_t \]

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

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

\[ B_0 \triangleq \text{in}_t.B_t + \text{in}_f.B_f \]

\[ B_t \triangleq \overline{\text{out}_t}.B_0 \]

\[ B_f \triangleq \overline{\text{out}_f}.B_0 \]
Linked boolean buffers

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

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

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

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

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

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

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

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

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

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

\[ \alpha \cup Q \cup \beta \]
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?

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

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

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

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

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

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

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

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

\[ \alpha \]

\[ \alpha \]

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

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

\[ \beta \]

\[ \beta \]

\[ \text{nil}|U \]

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

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

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

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

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

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

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

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

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

\[
\begin{array}{c}
\beta \\
\downarrow \\
\text{nil} | U
\end{array}
\quad
\begin{array}{c}
\beta \\
\downarrow \\
\text{nil} | \text{nil} | \beta . U
\end{array}
\]

\[
\begin{array}{c}
U \\
\downarrow \beta \\
\alpha | U
\end{array}
\quad
\begin{array}{c}
\beta \\
\downarrow \\
\alpha | \text{nil} | U
\end{array}
\]

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

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

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

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

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

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

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

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

\[ \alpha \quad \beta \]

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

\[ \beta \quad \alpha \quad \beta \quad \alpha \quad \beta \]

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

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

\[
\begin{array}{cccccccc}
U & \xrightarrow{\beta} & \alpha|U & \xrightarrow{\beta} & \alpha|\alpha|U & \xrightarrow{\beta} & \ldots \\
\text{\alpha} & & \text{\alpha} & & \text{\alpha} & & \text{\alpha} \\
\text{nil}|\beta.U & \xrightarrow{\alpha} & \alpha|\text{nil}|\beta.U & \xrightarrow{\beta} & \alpha|\text{nil}|U & \xrightarrow{\beta} & \ldots \\
\beta & & \text{\alpha} & & \beta & & \text{\alpha} \\
\text{nil}|U & & \text{nil}|\text{nil}|\beta.U & & \text{nil}|\text{nil}|U & & \ldots \\
\end{array}
\]
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: 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: LTS?

let’s ignore \texttt{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: \( inc, val, reset, stop \)

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.