Modelling and Validation of Concurrent System: the calculus of communicating systems (CCS)

António Ravara

May 6, 2024

Motivation

The "classical" models, according to Chomsky hierarchy

Operational / Denotational Models

1. Finite Automata / Regular Languages
   Represent finite-state systems
2. Push-Down Automata / Context-Free Languages
   Represent finite-state systems with a memory stack
3. Linear-Bounded Automata / Context-Sensitive Languages
   Represent finite-state systems with a finitely-long list as store
4. Turing Machines / Unrestricted Languages
   Represent finite-state systems with a infinitely-long list as store

The “classical” models, according to Chomsky hierarchy

Operational / Denotational Models

1. Finite Automata / Regular Languages
   Represent finite-state systems
2. Push-Down Automata / Context-Free Languages
   Represent finite-state systems with a memory stack
3. Linear-Bounded Automata / Context-Sensitive Languages
   Represent finite-state systems with a finitely-long list as store
4. Turing Machines / Unrestricted Languages
   Represent finite-state systems with a infinitely-long list as store

Isn’t this enough? Turing Machines are universal

Implement any computable function

One model per expressiveness class is enough, right?

If Turing Machines have all the power we need and are universal, why bother inventing other languages?

One model per expressiveness class is enough, right?

If Turing Machines have all the power we need and are universal, why bother inventing other languages?

Programming Languages

- Low / High level
- General purpose / DSLs
- Imperative / Functional / Logic
- Object-Oriented / Aspect-Oriented / Service-Oriented

One model per expressiveness class is enough, right?

If Turing Machines have all the power we need and are universal, why bother inventing other languages?

Programming Languages

- Low / High level
- General purpose / DSLs
- Imperative / Functional / Logic
- Object-Oriented / Aspect-Oriented / Service-Oriented

The intended system matters!

What aspects consider when modelling concurrent systems?

(Non-)Termination

- Sequential programs implement "functionalities"
  - One expects them to terminate and (sometimes) return a result
  - Examples: factorial, bank account, queue

What aspects consider when modelling concurrent systems?

(Non-)Termination

- Sequential programs implement “functionalities”
  - One expects them to terminate and (sometimes) return a result
  - Examples: factorial, bank account, queue
- Concurrent programs implement “behaviour”
  - One expects them to (often) run forever, being reactive and responsive
  - Examples: operating system, cloud storage, social network

What aspects consider when modelling concurrent systems?

Reactiveness is key

Sequential programs implement "functionalities"

- Run on demand, receiving input data and returning results
- Examples: factorial, bank account, queue

What aspects consider when modelling concurrent systems?

Reactiveness is key

Sequential programs implement "functionalities"

- Run on demand, receiving input data and returning results
- Examples: factorial, bank account, queue

Concurrent programs implement "behaviour"

- Are often idle (or with invisible activity), reacting to stimula
- Examples: ATM machine, sensor network, alarm system

How to consider...

Non-Termination

- Key aspect: accept infinite words
- (Simplest) Operational / Denotational Model
  Büchi Automata / ω-Regular Languages

How to consider...

Non-Termination

- Key aspect: accept infinite words
- (Simplest) Operational / Denotational Model
  Büchi Automata / ω-Regular Languages

Interaction

- Key aspects: communication and parallelism
- (Simplest) Operational / Denotational Model
  Calculus of Communicating Systems (CCS) / Labelled Transition Systems

Consider a vending machine

(a typical non-terminating reactive machine):

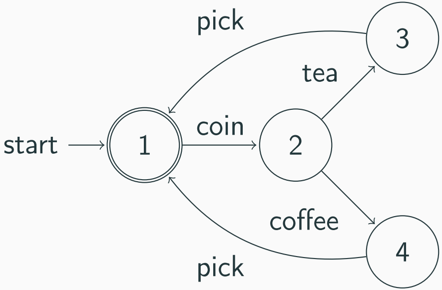
- Intended (user) behaviour:
  insert a coin; choose coffee or tea; pick the beverage

Modelling and reasoning about reactive systems

Consider a vending machine

(a typical non-terminating reactive machine):

- Intended (user) behaviour:
  insert a coin; choose coffee or tea; pick the beverage

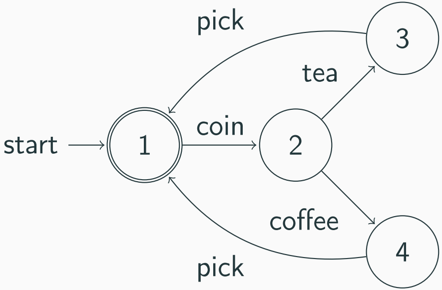


Modelling and reasoning about reactive systems

Consider a vending machine

(a typical non-terminating reactive machine):

- Intended (user) behaviour:
  insert a coin; choose coffee or tea; pick the beverage



- Denotational model: $coin.(\mathit{coffee} + \mathit{tea}).\mathit{pick}$

Modelling and reasoning about reactive systems

- A regular expression specifies the intended system behaviour
- An automaton implements the intended system behaviour

Modelling and reasoning about reactive systems

- A regular expression specifies the intended system behaviour
- An automaton implements the intended system behaviour

System Correctness: back to basics

The program (automaton) is *correct* if it implements (accepts) exactly the intended behaviour (the language of the regular expression).

- A regular expression specifies the intended system behaviour
- An automaton implements the intended system behaviour

System Correctness: back to basics

The program (automaton) is *correct* if it implements (accepts) exactly the intended behaviour (the language of the regular expression).

Is the vending machine correct?

- Language of the vending machine, by converting the automaton

$$coin.(coffee.pick + tea.pick)$$

- $coin.(coffee + tea).pick = coin.(coffee.pick + tea.pick)$

Is the vending machine correct?

- In the previous example we used the Kleene algebra distribution law (of sequencing over choice) to conclude the correctness of the automaton
  Kleene algebra transforms expressions preserving their language

Is the vending machine correct?

- In the previous example we used the Kleene algebra distribution law (of sequencing over choice) to conclude the correctness of the automaton
  Kleene algebra transforms expressions preserving their language
- The equivalence principle:
  two automaton / regular expression are equivalent if they accept / denote the same language

Is the vending machine correct?

- In the previous example we used the Kleene algebra distribution law (of sequencing over choice) to conclude the correctness of the automaton
  Kleene algebra transforms expressions preserving their language
- The equivalence principle:
  two automaton / regular expression are equivalent if they accept / denote the same language

$$coin.(coffee + tea).pick = coin.(coffee.pick + tea.pick)$$

So, the language of the automata is the same of that of the expression specifying the intended behaviour

Is the vending machine correct?

- In the previous example we used the Kleene algebra distribution law (of sequencing over choice) to conclude the correctness of the automaton
  Kleene algebra transforms expressions preserving their language
- The equivalence principle:
  two automaton / regular expression are equivalent if they accept / denote the same language

$$coin.(coffee + tea).pick = coin.(coffee.pick + tea.pick)$$

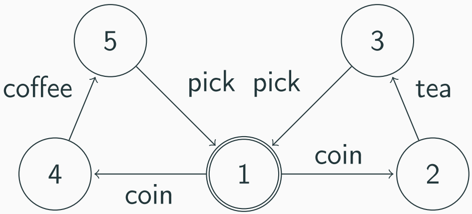
So, the language of the automata is the same of that of the expression specifying the intended behaviour

The vending machine is correct

Is the equivalence notion the right one for reactive systems?

An equivalent vending machine

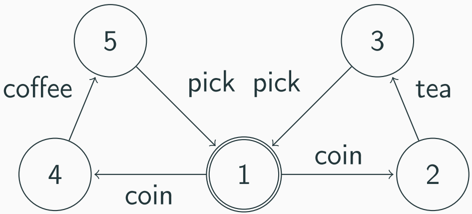
$coin.(coffee.pick + tea.pick) = coin.coffee.pick + coin.tea.pick$



Is the equivalence notion the right one for reactive systems?

An equivalent vending machine

$coin.(coffee.pick + tea.pick) = coin.coffee.pick + coin.tea.pick$



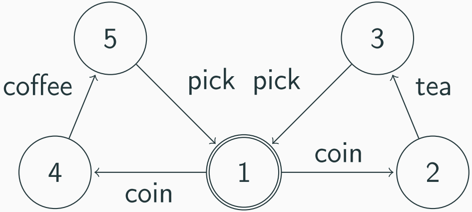
Does it have the same intended behaviour?

- When the user inserts the coin, the automaton non-deterministically decides to go to the left or to the right

Is the equivalence notion the right one for reactive systems?

An equivalent vending machine

$coin.(coffee.pick + tea.pick) = coin.coffee.pick + coin.tea.pick$



Does it have the same intended behaviour?

- When the user inserts the coin, the automaton non-deterministically decides to go to the left or to the right
- The user no longer can choose between tea or coffee...

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

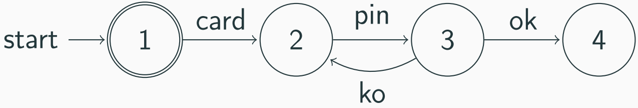
  An automata modelling it has final states?

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

  An automata modelling it has final states?

- Imagine the authentication phase of interacting with an ATM

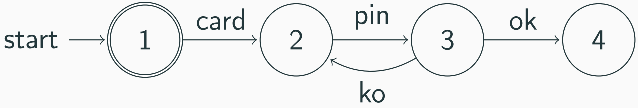


Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

  An automata modelling it has final states?

- Imagine the authentication phase of interacting with an ATM



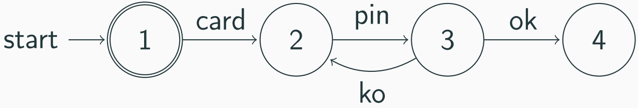
card and pin are user's actions; ok and ko are ATM's actions

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

  An automata modelling it has final states?

- Imagine the authentication phase of interacting with an ATM



card and pin are user's actions; ok and ko are ATM's actions

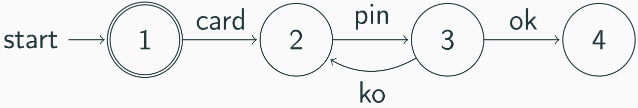
An automaton does not distinguish input from output actions

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

  An automata modelling it has final states?

- Imagine the authentication phase of interacting with an ATM

  card and pin are user's actions; ok and ko are ATM's actions

  An automaton does not distinguish input from output actions

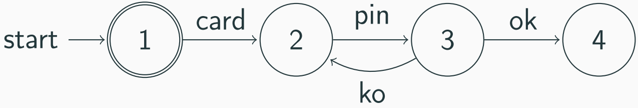
- Imagine a synchronous like VOIP
  The interacting parties need to communicate synchronously,
  with the input on one side match by output on the other

Are automata adequate to reactive systems?

- Imagine a streaming system like a internet TV channel
  It should never terminate

  An automata modelling it has final states?

- Imagine the authentication phase of interacting with an ATM



  card and pin are user's actions; ok and ko are ATM's actions

  An automaton does not distinguish input from output actions

- Imagine a synchronous like VOIP
  The interacting parties need to communicate synchronously, with the input on one side match by output on the other

  An automaton does not represent synchronous communication

Requirements for an adequate model of reactive systems

In short

- Represent non-terminating behaviour
  without considering necessarily final states
- Distinguish input and output actions
  and allow the input of one party to be the output of another
- Support parallelism and communication
  systems composed by (a)synchronous interactive components
  interacting with their environment
- Use a finer notion of equivalence
  taking choice into consideration

Requirements for an adequate model of reactive systems

In short

- Represent non-terminating behaviour
  without considering necessarily final states
- Distinguish input and output actions
  and allow the input of one party to be the output of another
- Support parallelism and communication
  systems composed by (a)synchronous interactive components
  interacting with their environment
- Use a finer notion of equivalence
  taking choice into consideration

What is an appropriate operational / denotational model?

Calculus of Communicating Systems (CCS)

CCS: a process algebra

- Syntax: a language defined by a regular grammar
- Operational semantics: a transition relation
- Denotational semantics: a mathematical representation of non-terminating reactive systems

CCS: a process algebra

- Syntax: a language defined by a regular grammar
- Operational semantics: a transition relation
- Denotational semantics: a mathematical representation of non-terminating reactive systems

Mathematical interpretation

Labelled Transition Systems, equipped with a congruence relation

Calculus of Communicating Systems (CCS)

CCS: a process algebra

- Syntax: a language defined by a regular grammar
- Operational semantics: a transition relation
- Denotational semantics: a mathematical representation of non-terminating reactive systems

Mathematical interpretation

Labelled Transition Systems, equipped with a congruence relation

Congruence

Substitutive equivalence preserved by the operations of the language

Calculus of Communicating Systems (CCS)

Why *calculus*?

A minimal mathematical language for calculations and reasoning

Calculus of Communicating Systems (CCS)

Why *calculus*?

A minimal mathematical language for calculations and reasoning

Examples:

- Leibniz's infinitesimal calculus
- Newton's integral calculus

Why *calculus*?

A minimal mathematical language for calculations and reasoning

Examples:

- Leibniz's infinitesimal calculus
- Newton's integral calculus

Why *minimal*?

Ockham's razor Principle: *lex parsimoniae*

Calculus of Communicating Systems (CCS)

Why *calculus*?

A minimal mathematical language for calculations and reasoning

Examples:

- Leibniz's infinitesimal calculus
- Newton's integral calculus

Why *minimal*?

Ockham's razor Principle: *lex parsimoniae*

Shaves off unnecessary hair – the best definition/explanation is the simplest one

Basically, what is a reactive system?

A process able of performing (interactive) actions and after each one, becoming another process

Motto (Tony Hoare and Robin Milner)

(In reactive systems) Everything is a process!

Remember set theory? In Mathematics, everything is a set

Calculus of Communicating Systems (CCS)

Assume a countable set $\mathcal{N}$ of action *names*; then CCS actions are defined as follows:

$\alpha ::=$		Actions
	a	Input action
	$\mid \overline{a}$	Output action
	$\mid \tau$	Silent (internal) action

a and $\overline{a}$ are observable actions, while τ is an unobservable action.

Calculus of Communicating Systems (CCS)

Assume a countable set $\mathcal{N}$ of action *names*; then CCS actions are defined as follows:

$$
\begin{array}{rr}
\alpha ::= & \text{Actions} \\
a & \text{Input action} \\
\mid \overline{a} & \text{Output action} \\
\mid \tau & \text{Silent (internal) action}
\end{array}
$$

a and $\overline{a}$ are observable actions, while τ is an unobservable action.

Processes

A computing agent able of performing internal computation and of interacting with its environment via communicating actions.

Syntax of CCS

Processes

Consider for each process variable A a defining equation $A(x_1, \ldots, x_n) = P$ where the name variables $x_1, \ldots, x_n$ occur (bound) in P.

Syntax of CCS

Processes

Consider for each process variable A a defining equation $A(x_1, \ldots, x_n) = P$ where the name variables $x_1, \ldots, x_n$ occur (bound) in P.

$$
\begin{array}{llr}
P, Q, R ::= & & \text{Processes} \\
& 0 & \text{Empty process} \\
& \mid A\langle a_1, \ldots, a_n \rangle & \text{process definition} \\
& \mid \alpha.P & \text{action prefix} \\
& \mid (\mathbf{new}\, a)P & \text{action hiding} \\
& \mid P \mid Q & \text{parallel composition} \\
& \mid P + Q & \text{(non-deterministic) choice}
\end{array}
$$

Ingredients of CCS

- Actions are
  - *offers* (inputs)
  - *selections* (outputs),
  - or *idle* (invisible)
- Hiding makes actions invisible
- Parallel composition allows synchronous (by handshake) communication between two processes
- Definitions support generic processes and recursion

Ingredients of CCS

- Actions are
  - *offers* (inputs)
  - *selections* (outputs),
  - or *idle* (invisible)
- Hiding makes actions invisible
- Parallel composition allows synchronous (by handshake) communication between two processes
- Definitions support generic processes and recursion

Precedence in decreasing order

- hiding
- prefixing
- parallel composition
- choice

Vending Machine

In a process equation $A(x_1, \ldots, x_n)$ or definition $A\langle a_1, \ldots, a_n \rangle$, if $n = 0$ we simply write A

Version 1

- $VM = coin.(tea.\overline{pick}.VM + coffee.\overline{pick}.VM)$
- $Client = \overline{coin}.\overline{coffee}.pick.0$
- $System = VM \mid Client$

Vending Machine

In a process equation $A(x_1, \ldots, x_n)$ or definition $A\langle a_1, \ldots, a_n \rangle$, if $n = 0$ we simply write A

Version 1

- $VM = coin.(tea.\overline{pick}.VM + coffee.\overline{pick}.VM)$
- $Client = \overline{coin}.\overline{coffee}.pick.0$
- $System = VM \mid Client$

Version 2

- $VM = coin.\overline{sugar}.(yes.fill.Serve + no.Serve)$
- Serve $= (tea.\overline{pick}.VM + coffee.\overline{pick}.VM)$

ATM

- $Box = card.Session$
- $Session = (pin.(\overline{ok}.Serve + \overline{ko}.Session) + exit.\overline{card}.Box)$
- $Serve = (balance.\overline{pick}.Session + deposit.amount.\overline{pick}.Session + withdraw.amount.\overline{pick}.Session)$
- $Client = \overline{card}.\overline{pin}.Use$
- $Use = (ok.\overline{balance}.pick.\overline{pin}.ok.\overline{withdraw}.\overline{Fifty}.pick.\overline{exit}.card.0 + ko.\overline{exit}.card.0)$
- $System = Box \mid Client$

Actions, free and bound actions, and action substitution

Actions of a process $\mathsf{Act}(P) \subseteq \mathsf{Act}$

is a set inductively defined by the following rules.

$$\begin{aligned}
\mathsf{Act}(A\langle a_1, \ldots, a_n \rangle) &= \{a_1, \ldots, a_n\} \\
\mathsf{Act}(\alpha.P) &= \{a\} \cup \mathsf{Act}(P), \ \text{if } \alpha = a \text{ or } \alpha = \overline{a} \\
\mathsf{Act}((\mathbf{new}\, a)P) &= \{a\} \cup \mathsf{Act}(P) \\
\mathsf{Act}(P \mid Q) &= \mathsf{Act}(P) \cup \mathsf{Act}(Q) \\
\mathsf{Act}(P + Q) &= \mathsf{Act}(P) \cup \mathsf{Act}(Q)
\end{aligned}$$

Actions, free and bound actions, and action substitution

Free and bound actions

- $\mathsf{fn}(P) = \mathsf{Act}(P) \setminus \mathsf{bn}(P)$
- $\mathsf{bn}(P) \subseteq \mathsf{Act}$ is a set inductively defined by the rules

$$
\begin{aligned}
\mathsf{bn}(A\langle a_1, \ldots, a_n \rangle) &= \emptyset \\
\mathsf{bn}(\alpha.P) &= \mathsf{bn}(P) \\
\mathsf{bn}((\mathbf{new}\, a)P) &= \{a\} \cup \mathsf{bn}(P) \\
\mathsf{bn}(P \mid Q) &= \mathsf{bn}(P) \cup \mathsf{bn}(Q) \\
\mathsf{bn}(P + Q) &= \mathsf{bn}(P) \cup \mathsf{bn}(Q)
\end{aligned}
$$

Actions, free and bound actions, and action substitution

Substitution

Let $P\{\vec{a} \leftarrow \vec{b}\}$ denote the simultaneous substitution of the free occurrences of the actions $\vec{a}$ in P for $\vec{b}$.

Example

$$
\begin{aligned}
\{water \leftarrow coffee\}\{cola \leftarrow tea\}(tea.\overline{pick}.VM + coffee.\overline{pick}.VM) \\
\{water \leftarrow coffee\}(cola.\overline{pick}.VM + coffee.\overline{pick}.VM) \\
(cola.\overline{pick}.VM + water.\overline{pick}.VM)
\end{aligned}
$$

Actions, free and bound actions, and action substitution

It is sometimes necessary to rename bound actions to avoid clashes.

$$((\textbf{new}\, a)a.b.0)\{a \leftarrow b\} = (\textbf{new}\, a)a.a.0$$

The free action b became a, which is bound...

Alpha-congruence

The binary relation $=_\alpha$ on processes is inductively defined by the rule $(\textbf{new}\, a)P =_\alpha (\textbf{new}\, b)P\{a \leftarrow b\}$ if $b \notin \text{bn}(P)$, and homomorphic rules on the remaining process constructs.

Structural Operational Semantics of CCS

- Syntax-driven proof rules to infer the behaviour of a system (Gordon Plotkin, 1981)
- Rules describe single computational steps, explaining the effect of executing a particular (syntactic) construct of the language

Transition Relation

Given a set of CCS defining equations $\mathcal{P}$ specifying a system, the transition relation of the system is defined by a set of triples

$$\{\xrightarrow{a} \in \mathcal{P} \times \mathcal{P} \mid a \in \mathsf{Act}\}$$

In the next slide we inductively define the Structural Operational Semantics of CCS

SOS proof rules of CCS

$$\frac{P_A\{\vec{a}\leftarrow\vec{b}\} \xrightarrow{\alpha} P'}{A\langle\vec{b}\rangle \xrightarrow{\alpha} P'}\ A(\vec{a}) \stackrel{\text{def}}{=} P_A\ \text{[Def]} \qquad \frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{[Pre]}$$

$$\frac{P \xrightarrow{\alpha} P'}{(\textbf{new}\,a)P \xrightarrow{\alpha} (\textbf{new}\,a)P'}\ \alpha \notin \{a, \overline{a}\}\ \text{[Res]} \qquad \frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} P'}\ P =_\alpha Q\ \text{[Alpha]}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{(Q \mid P) \xrightarrow{\alpha} (Q' \mid P)}\ \text{[L-Par]} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \text{[L-Sum]}$$

$$\frac{Q \xrightarrow{a} Q' \quad P \xrightarrow{\overline{a}} P'}{(Q \mid P) \xrightarrow{\tau} (Q' \mid P')}\ \text{[L-Sync]}$$

Par, Sum and Sync have right rules.

Running one example

Recall that

$$Serve = (tea.\overline{pick}.VM + coffee.\overline{pick}.VM)$$

and $VM \mid Client =$

$$coin.\overline{sugar}.(yes.fill.Serve + no.Serve)|\overline{coin}.sugar.\overline{no}.\overline{coffee}.pick.0$$

So,

$$\begin{aligned}
VM|Client &\xrightarrow{\tau} \overline{sugar}.(yes.fill.Serve + no.Serve)|sugar.\overline{no}.\overline{coffee}.pick.0\\
&\xrightarrow{\tau} (yes.fill.Serve + no.Serve) \mid \overline{no}.\overline{coffee}.pick.0\\
&\xrightarrow{\tau} Serve \mid \overline{coffee}.pick.0\\
&\xrightarrow{\tau} \overline{pick}.VM \mid pick.0\\
&\xrightarrow{\tau} VM \mid 0
\end{aligned}$$

How to justify each step?

Steps 4 and 5 are direct applications of the [Sync] rule.

Step 1

Let $Decide = \overline{sugar}.(yes.fill.Serve + no.Serve)$ and
$BCoffee = sugar.\overline{no}.\overline{coffee}.pick.0$

$$
\dfrac{
\dfrac{\dfrac{}{coin.Decide \xrightarrow{coin} Decide}\,[\text{Pre}]}{VM \xrightarrow{coin} Decide}\,[\text{Def}]
\qquad
\dfrac{\dfrac{}{coin.BCoffee \xrightarrow{\overline{coin}} BCoffee}\,[\text{Pre}]}{Client \xrightarrow{\overline{coin}} BCoffee}\,[\text{Def}]
}{VM \mid Client \xrightarrow{\tau} Decide \mid BCoffee}\,[\text{Sync}]
$$

Step 2 is similar to step 1

How to justify each step?

Step 3

Let $\mathit{Coffee} = \overline{\mathit{coffee}}.\mathit{pick}.0$

$$\dfrac{\dfrac{\dfrac{}{\mathit{no}.\mathit{Serve} \xrightarrow{\mathit{no}} \mathit{Serve}}\,[\mathrm{Pre}]}{\mathit{yes}.\mathit{fill}.\mathit{Serve} + \mathit{no}.\mathit{Serve} \xrightarrow{\mathit{no}} \mathit{Serve}}\,[\mathrm{Sum}] \qquad \dfrac{}{\overline{\mathit{no}}.\mathit{Coffee} \xrightarrow{\overline{\mathit{no}}} \mathit{Coffee}}\,[\mathrm{Pre}]}{\mathit{yes}.\mathit{fill}.\mathit{Serve} + \mathit{no}.\mathit{Serve} \mid \overline{\mathit{no}}.\mathit{Coffee} \xrightarrow{\tau} \mathit{Serve} \mid \mathit{Coffee}}\,[\mathrm{Sync}]$$