The π-calculus

So far we explored, using CCS, the concepts of ① INTERACTION, ② BEHAVIOURAL EQUIVALENCE, ③ CONCURRENCY & COMMUNICATION. Now we investigate ④ MOBILITY with the help of an extension of CCS: the π-calculus, which was introduced with the explicit intent of modelling mobility, an aspect of concurrent systems that is not captured by CCS.

What is mobility?

Let us explain the concept using an example. Consider a system consisting of a server S and an arbitrary number of clients $C_1, \dots, C_m$. The server accepts incoming connections from clients. In CCS we would model this with an action, say 'connect'. After connection the server is supposed to have a private conversation with the client C_i which initiated the connection. The problem is that if S and C_i use actions that use free names, any other C_j may be able to interfere with the ongoing conversation between S and C_i

$$S := \text{connect}.\text{talk}.S'$$
$$C_1 := \overline{\text{connect}}.\overline{\text{talk}}.C_1'$$
$$C_2 := \overline{\text{talk}}.C_2'$$

$$S \parallel C_1 \parallel C_2 \longrightarrow \text{talk}.S' \parallel \overline{\text{talk}}.C_1' \parallel \overline{\text{talk}}.C_2'$$
$$\longrightarrow S' \parallel \overline{\text{talk}}.C_1' \parallel C_2'$$

C_1 initiated the connection but C_2 interfered

One can try to fix it by creating separate $\text{connect}_i, \text{talk}_i$ actions for each client but this only works if one has a fixed number of clients. A more promising direction is creating a restricted name to be used for communication that needs to be private between S and C_i

$$C_i := \overline{\text{connect}}.\nu\text{talk}.(\overline{\text{talk}}.C_i')$$

but now we have a problem: no process, but the ones in the static scope of νtalk (i.e. in the continuation of $\overline{\text{connect}}$), will ever be able to acquire knowledge of this new name. We would need to transfer it to the server.

The essence of mobility is the ability to acquire knowledge of a name through communication. In the π-calculus this is modelled by letting action prefixes have arguments: input actions can now have arguments that will be bound to the "contents" of the message sent through the main action in this context it is useful to think of a name as a channel that can be used to transmit messages. CCS actions are the corner case in which messages have no content.
Similarly, output actions will also indicate the names to be sent as messages to the receiver.

Notationally, the receiver is a process $a(x).P$, waiting on the "channel" a for a message m to arrive. When the message arrives it is bound to the variable x, and the process continues as $P[^m/_x]$.

The sender is a process $\overline{a}\langle m\rangle.Q$ that is able to synchronise with a receiver as above by transmitting the message m through the channel a and then continue as Q.
The synchronisation on a channel is synchronous so the sender can "execute" an output $\overline{a}\langle m\rangle$ only when another process is ready to "executing" a corresponding input $a(x)$. The message exchanged is itself a name, which can be used later on as a channel for synchronisation. This use of names is what makes the π-calculus at the same time very expressive and very succinct.
To complete our server/clients example, the protocol can be modelled as

$$S = connect(x).x.S'$$
$$C = \nu talk.\ \overline{connect}\langle talk\rangle.\overline{talk}.C'$$

$$S \parallel C \parallel C \longrightarrow \nu talk.(talk.S' \parallel \overline{talk}.C') \parallel C \longrightarrow \nu talk\,(S' \parallel C') \parallel C$$

the private name has been acquired by the server ($x \mapsto talk$)

cannot synch on talk because it cannot know it

Syntax of π-calculus

$$P ::= \mathbf{0} \mid P \| P' \mid \nu z.P \mid A[\vec{z}] \mid \underbrace{\pi_1.P_1 + \ldots + \pi_m.P_m}_{\text{sums ranged over by } M, N} \quad \text{(sequential procs.)}$$

$$\pi ::= \underbrace{x(\vec{y})}_{\text{INPUT prefix}} \mid \underbrace{\bar{x}\langle\vec{y}\rangle}_{\text{OUTPUT prefix}} \mid \tau$$

Terminology

- In a process $\pi.P$, P is called the continuation of the prefix π
- In $x(\vec{y})$ and $\bar{x}\langle\vec{y}\rangle$, $\vec{y}$ is a tuple of names.
  They are all distinct in input prefixes (because they are bound, see below)
  When we allow only prefixes with:
  - $|\vec{y}| = 0$ then the calculus coincides with CCS
  - $|\vec{y}| = 1$ then we speak of "monadic π-calculus"
  - $|\vec{y}| \geq 0$ (the general case) we speak of "polyadic π-calculus"

- When a prefix has no arguments, we omit the brackets: x is $x()$, $\bar{x}$ is $\bar{x}\langle\rangle$
- For a prefix $x(\vec{y})$ or $\bar{x}\langle\vec{y}\rangle$, x is said to be the subject and $\vec{y}$ the object of the prefix
- As for CCS we abbreviate $\pi.\mathbf{0}$ with π
- We abbreviate $\nu z_1.\nu z_2.\cdots\nu z_m.P$ with $\nu\vec{z}.P$

Binding

Both $x(\vec{z}).P$ and $\nu\vec{z}.P$ bind $\vec{z}$, with scope P.

Occurrences of names of $\vec{z}$ in P are bound, and free if not bound.

For any P, $bn(P)$ is the set of names bound in some subterm of P, $fn(P)$ is the set of names occurring free in P. A process P with $fn(P) = \emptyset$ is said "closed".

Example $P = x(y).\nu z.\bar{y}\langle z, z\rangle \| \nu y'.\bar{x}\langle y'\rangle.y'(u, w)$

$fn(P) = \{x, z\}$ $\quad bn(P) = \{y, z, y', u, w\}$

We still make the NO-CLASH Assumption:

NO-CLASH: $fn(P) \cap bn(P) = \emptyset$ and no name is bound again in the same scope, that is to say that in terms $\nu x.P$ or $a(\vec{z}).P$ with $x \in \vec{z}$, NO restriction or prefix can bind x in P.

We can always use α-conversion (i.e. renaming of a bound name to a fresh one) to disambiguate a term so it meets the NO-CLASH assumption.

Example: $\nu y.\overline{x}\langle y\rangle.a(x,z).((\nu z.\overline{x}\langle z\rangle) \parallel \overline{z}\langle a\rangle) = P'$

the arrows point a bound occurrence to its binder.
Note that x is a free name of P but is also bound in one of its subterms. To meet NO-CLASH we used α-conversion

$$P' \equiv_\alpha \nu y.\overline{x}\langle y\rangle.a(x',z).((\nu z'.\overline{x'}\langle z'\rangle) \parallel \overline{z}\langle a\rangle) = P$$

Now $fn(P) = \{x, a\}$, $bn(P) = \{y, x', z, z'\}$ which are disjoint and we avoided the nested binding of z.
On the other hand the term $(\nu x.\overline{a}\langle x\rangle) \parallel (\nu x.\overline{a}\langle x\rangle)$ meets the NO-CLASH assumption because the two binders have disjoint scopes.

Note: α-conversion CANNOT rename free names!

Definitions As for CCS, the behaviour of process instances $A[\vec{a}]$ is specified by means of a set of definitions Δ of the form

$$A[\vec{x}] := \sum_{i \in I} \pi_i.P_i \quad \text{where } fn\left(\sum_{i \in I} \pi_i.P_i\right) \subseteq \vec{x}$$

We still assume Δ contains at most one definition for each process identifier $A \in PID$.

In the π-calculus, we always assume

Δ is finite.

While the presented syntax is convenient for writing examples, we also introduce a restriction on syntax that makes the presentation of formal arguments more concise.

Def A set of definitions Δ is said to be <u>normalised</u> if every definition $(A[\vec{x}] := \sum_{i \in I} \pi_i . P_i) \in \Delta$ is such that P_i contains no sum, for all $i \in I$.
Similarly a process P is <u>normalised</u> if it contains no sums.

It is easy to see that any program P, Δ can be normalised:

$\nu x . (A[x,y] \parallel \overbrace{\overline{x}\langle y \rangle}^{\text{a sum!}})$

$A[x,y] := x(z).\underbrace{\overline{z}\langle y \rangle}_{\text{a sum!}}$

NOT NORMALISED

$\rightsquigarrow$

$P = \nu x . (A[x,y] \parallel B[x,y])$

$A[x,y] := x(z).C[y,z]$

$B[x,y] := \overline{x}\langle y \rangle$

$C[y,z] := \overline{z}\langle y \rangle$

NORMALISED

<u>Contexts</u> are defined, modulo prefixes, as in CCS:

$$C ::= [\,] \mid \pi.C + M \mid \nu a.C \mid C \parallel P \mid P \parallel C$$

The elementary contexts are $\pi.[\,] + M$, $\nu a.[\,]$, $[\,] \parallel P$, $P \parallel [\,]$.
A process congruence is an equivalence that is preserved by all elementary contexts.
As for CCS we still have that a congruence is preserved by all contexts.

<u>Substitutions</u> work as in CCS and are assumed to respect the NO-CLASH assumption

<u>Structural Congruence</u> $\equiv$ is straightforwardly extended to π-calculus:
The relation $\equiv$ is the smallest congruence including the commutativity and associativity laws for $\parallel$, $+$, α-conversion and satisfies:

$0 \parallel P \equiv P$

$\nu a.0 \equiv 0$

$\nu x.\nu y.P \equiv \nu y.\nu x.P$ EXCHANGE

$\nu x.(P \parallel Q) \equiv P \parallel \nu x.Q$ if $x \notin fn(Q)$ SCOPE EXTRUSION

We now formalise the internal interaction between components with the reaction transition $\longrightarrow$ defined by the rules:

REACTION RULES

$$\text{REACT}: \quad \frac{}{(\overline{x}\langle\vec{y}\rangle.P_1 + M_1) \parallel (x(\vec{z}).P_2 + M_2) \longrightarrow P_1 \parallel P_2\left[\vec{y}/\vec{z}\right]} \; |\vec{y}| = |\vec{z}|$$

$$\text{TAU}: \quad \tau.P + M \longrightarrow P \qquad \text{PAR}: \quad \frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$$

$$\text{RES}: \quad \frac{P \longrightarrow P'}{\nu a.P \longrightarrow \nu a.P'} \qquad \text{STRUCT}: \quad \frac{P_1 \equiv_\Delta P_2 \longrightarrow P_2' \equiv_\Delta P_1'}{P_1 \longrightarrow P_1'}$$

Here $\equiv_\Delta$ is the usual extension of $\equiv$ that satisfies $A[\vec{a}] \equiv_\Delta Q\left[\vec{a}/\vec{x}\right]$ if $(A[\vec{x}] := Q) \in \Delta$

Example We present a complete server/clients system (normalised!)

$$S[s] := s(x).(\nu d.\ \overbrace{A[x,d]}^{\text{answer to the client}} \parallel S[s])$$

- s: the "address" of the server
- x: waiting for a request
- d: address of the requesting client
- $A[x,d]$: new data we want to send to the client
- $S[s]$: the server continues asynchronously without waiting for the client to receive the answer

$A[x,d] := \overline{x}\langle d\rangle$ the process sending the answer to the client x

$$C[s,m] := \tau.(\underbrace{Q[s,m]}_{\text{request to server holding the address } m} \parallel \underbrace{C'[s,m]}_{\text{the client asynchronously waits for an answer}})$$

- s: address of server
- m: address of client's own mailbox

$$Q[s,m] := \overline{s}\langle m\rangle$$

$$C'[s,m] := \underbrace{m(y)}_{\text{wait for data } y \text{ coming to mailbox } m}.\ \underbrace{C[s,m]}_{\text{data is consumed internally, discarded and we go back to the start}}$$

A server with many clients can be modelled by the process $\nu s.(S[s] \parallel \nu m_0.C[s,m_0] \parallel \cdots \parallel \nu m_n.C[s,m_n])$

Note how each client has its own private mailbox address m_i.

Since we want to model the fact that clients may join the system at random we define an 'environment' process

$$E[s] := \tau.(\nu m.C[s,m] \parallel E[s])$$

Let us explore some reductions from $\nu s.(S[s] \parallel E[s])$

$$\nu s.(S[s] \parallel E[s]) \longrightarrow \nu s.(S[s] \parallel \nu m\, C[s,m] \parallel E[s])$$

$$\longrightarrow \nu s.(S[s] \parallel \nu m.C[s,m] \parallel \nu m'.C[s,m'] \parallel E[s])$$

$$\longrightarrow \nu s.(S[s] \parallel \nu m.(Q[s,m] \parallel C'[s,m]) \parallel \nu m'.C[s,m'] \parallel E[s])$$

$$\equiv_{\Delta} \nu s.(s(x).S' \parallel \nu m.(\bar{s}\langle m\rangle \parallel C'[s,m]) \parallel \nu m'.C[s,m'] \parallel E[s])$$

by SCOPE EXTRUSION $\equiv \nu s.(\nu m.(s(x).S' \parallel \bar{s}\langle m\rangle \parallel C'[s,m]) \parallel \nu m'.C[s,m'] \parallel E[s])$

$$\longrightarrow \nu s.(\nu m.\nu d.(\bar{m}\langle d\rangle \parallel m(y).C[s,m]) \parallel S[s] \parallel \nu m'.C[s,m'] \parallel E[s]) = P$$

$$\longrightarrow \nu s.(\nu m.\nu d.\, C[s,m] \parallel S[s] \parallel \nu m'.\, C[s,m'] \parallel E[s])$$

The reader should note two things:

1) Using SCOPE EXTRUSION at the marked step is absolutely essential to be able to apply the substitution $[^m/_x]$ binding the correct m to x. This is in stark contrast with CCS where scope extrusion was not essential to the flow of information.

2) From the intermediate process P the second client $C[s,m']$ can make a τ transition but its request can in no way interfere with the answer of the server to the first client i.e. the second client cannot steal the data sent to the first

A remark on the modelling power of names.

In the π-calculus there are only two kinds of entity: Processes and Names. As we have partially seen, names can model a number of concepts and lend themselves to many interpretations. They can model:

- Addresses (like IP address + port or locations)
- Communication channels / Links
- Memory references (pointers – but without pointer arithmetics)
- Abstraction of data
- Encryption keys / nonces (see the Spi-calculus for more on this)

... and more!

[Just to give you an idea of the variety of systems that can be fruitfully modelled by the π-calculus, let us mention that biologists use process algebra to study biological processes – there is even a full model of the living cell in π-calculus!]

The reason for the flexibility of names is that they possess some essential properties common to many entities, but not more: they are dynamically created unique atoms.

- dynamically: You can create them during execution
- unique: every time you create one it is guaranteed to be distinct from the others
- atoms: they are not structured and cannot be inspected. The only relevant property is whether they are the same as some other name or not.

[For more on the nature and properties of names, an interesting framework for their study is the one of Nominal Sets.]

Let us now introduce two useful representations of processes.

Def A process is in STANDARD FORM if it is in the form

$$\nu x_1 \nu x_2 \ldots \nu x_m . (Q_1 \parallel \ldots \parallel Q_k)$$

where each Q_i for $i = 1, \ldots, k$ is a (non zero) sequential process and each x_j for $j = 1, \ldots, n$ occurs free in some Q_i.

Theorem Every process P is structurally congruent to a standard form. This standard form is unique up to α-conversion, reordering of parallel components and of restrictions, so we denote it by $std(P)$.

Proof Same as for CCS, see EXERCISE SHEET 3, Problem 2.

Notation When we write $std(P) = \nu \vec{x} . Q$ we are asserting that Q is a parallel composition of non-zero sequential processes.

Note that when P is normalised, $std(P) = \nu \vec{x} . (A_1[\vec{a}_1] \parallel \ldots \parallel A_k[\vec{a}_k])$ i.e. all the sequential subterms of $std(P)$ are process instances.

Example Take the process P from the server/client example

$$P \equiv \nu s . (\nu m . \nu d . (A[m,d] \parallel C'[s,m]) \parallel S[s] \parallel \nu m . C[s,m] \parallel E[s])$$

$$\equiv \nu s . \nu m_1 . \nu m_2 . \nu d . (A[m_1,d] \parallel C'[s,m_1] \parallel S[s] \parallel C[s,m_2] \parallel E[s]) = std(P)$$

We now formalise, starting from standard form, the concept of COMMUNICATION TOPOLOGY (a.k.a. NETWORK) which is a graphical representation of processes that makes explicit what are the current links between the sequential processes.

Def Let P be a process with $std(P) = \nu x_1 \dots x_n . (Q_1 \| \dots \| Q_k)$.

The COMMUNICATION TOPOLOGY of P is the bipartite labelled graph $G[\![P]\!] = (S, N, E, \lambda)$ where

- $S = \{1, \dots, k\}$ is the set of nodes associated with the seq. proc. $Q_1 \dots Q_k$
- $N = \{x_1, \dots, x_n\}$ is the set of nodes associated with the restrictions $\nu x_1 \dots x_n$
- $E = \{(i, x_j) \mid x_j \in fn(Q_i)\}$
- $\lambda : S \uplus N \longrightarrow N \cup \{Q_1, \dots, Q_k\}$ with $\lambda(i) = Q_i$ $i \in S$, $\lambda(x_j) = x_j \in N$

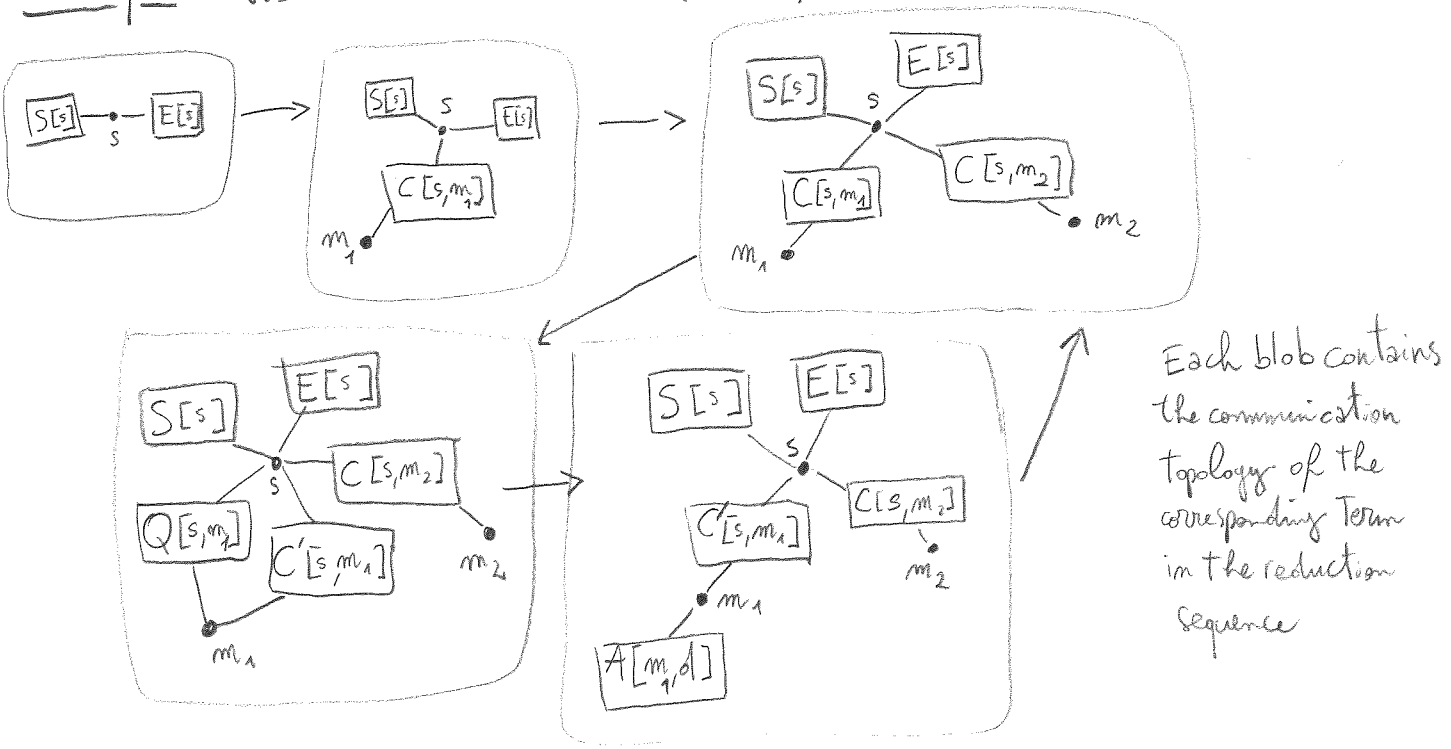
Note that $G[\![P]\!]$ is unique up to α-conversion and reordering of nodes in the same way $std(P)$ is unique up to $\equiv$.

Notation We will draw nodes in S with rectangles containing the corresponding label $\boxed{Q_i}$ while nodes in N will be drawn as dots $\bullet\, x_j$

Remark An equivalent way to formalise $G[\![P]\!]$ is by using hypergraphs with elements of S as hyperedges and E as incidence matrix.

Note that $G[\![P]\!]$ contains all the information needed to reconstruct P from the graph itself.
Note also that free names of P do not give rise to nodes in $G[\![P]\!]$, they only occur in labels!

Example We draw the reaction sequence for the server/client example

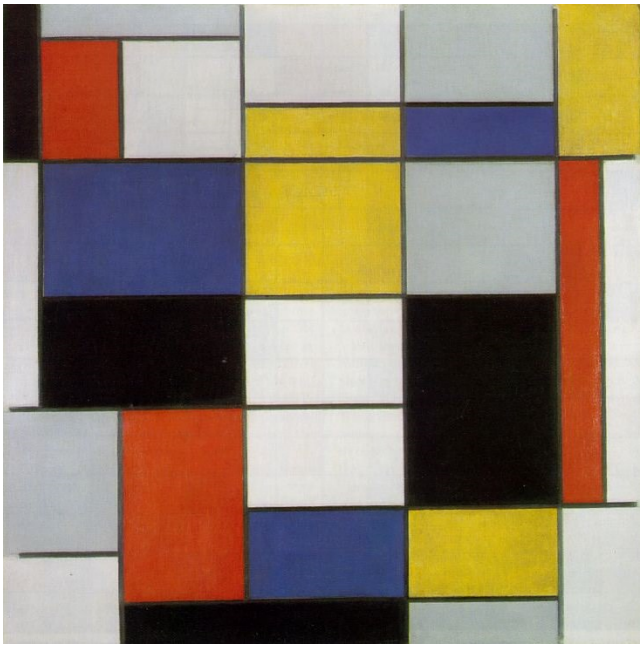


Each blob contains the communication topology of the corresponding Term in the reduction sequence

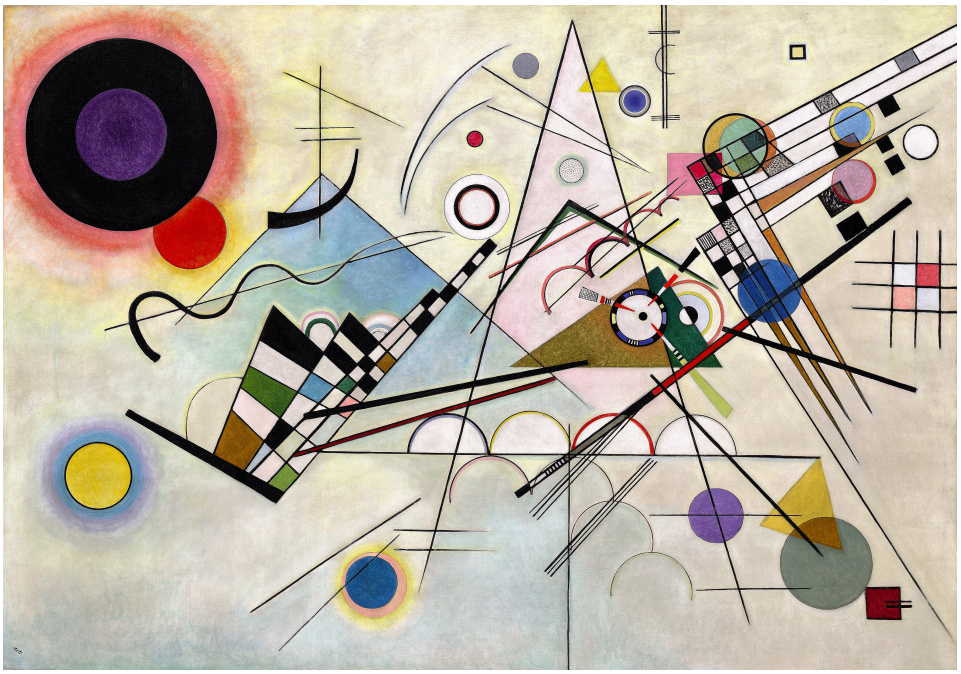
Mobility is what makes the evolution of the Comm Top. particularly complicated (and interesting!): the connectivity of the processes is highly dynamic in that processes can 'disconnect' or 'connect' with each other by exchanging names through known channels.

This cannot happen in CCS: new names and new processes can be created dynamically but these names will never give rise to new connection between pre-existing processes.

In a way, the difference between CCS topologies and π-calculus is analogous to the difference between a Mondrian painting and a Kandinsky:



Piet Mondrian
Composition Number 2



Wassily Kandinsky
Composition VIII

The Mondrian is structured in well defined areas that are further subdivided into other areas and so on. There is no "penetration" of one area into another. It is the same for CCS scopes.

The Kandinsky looks much more "organic" and dynamic (in fact K. took inspiration from microscope images of microorganisms for his later production!) where islands of CCS-like grids are penetrated and interconnected by filaments, irregular shapes and color.

Variants of π-calculus

A rich field of study in process algebra is the one that investigates the relative expressiveness of different sets of "primitives" to describe a system. We consider only some simple examples.

MONADIC vs POLYADIC π-calculus

Suppose only a single name can be sent over a channel at a time. Does this limit the expressivity of the language?
To answer this question we can show how to implement polyadic prefixes only using monadic prefixes. The encoding is a nice application of mobility. Assume $w \notin fn(P)$.

$$x(y_1, \dots, y_n).P \quad \rightsquigarrow \quad x(w).w(y_1).\dots.w(y_n).P$$

$$\overline{x}\langle a_1, \dots, a_n\rangle.P \quad \rightsquigarrow \quad \nu w.\overline{x}\langle w\rangle.\overline{w}\langle a_1\rangle.\dots.\overline{w}\langle a_n\rangle.P$$

The encoding uses the private channel w to prevent other components to interfere midway with the monadic encoding.
This encoding is correct provided all names are used uniformly by the processes: while $\overline{x}\langle a,b,c\rangle.P \parallel x(y,z).Q \not\to$ in the polyadic calculus because the "arities" of the prefixes do not match, the encoding can make the receiver continue after a and b have been communicated!

For the interested reader: enforcing a coherent use of names in the polyadic π-calculus is done by means of a technique called "SORTING". You can interpret these as assigning types to names. Infact there are type systems that can check if a π-calculus term respects a SORTING.

REPLICATION vs RECURSIVE DEFINITIONS

A very common alternative presentation of π-calculus avoids using definitions completely and provides a new operator to express infinite behaviour: the replication (aka "bang") operator $!P$. The syntax of the π-calculus with replication is:

$$P ::= 0 \mid \sum_{i \in I} \pi_i . P_i \mid P \| P \mid \nu a . P \mid !P$$

Note the absence of process instances and thus of definitions. The only extension we need to make on our definitions to explain the behaviour of $!P$ is stating that $![\,]$ is an elementary context and that structural congruence also satisfies the REPLICATION LAW

$$!P \equiv P \| !P$$

Note that this, in conjuction with the STRUCT reaction rule, gives a specification for the reaction semantics of replication. For example, the "environment" process definition $E[s]$ of the server example can instead be implemented with the term $!(\nu m . C[s,m])$. (Note that the original definition required an extra τ action to spawn a new client).

Similarly, the server process could be expressed with

$$!\left(s(x) . \nu d . \; \bar{x}\langle d \rangle\right)$$

A natural question is: is replication powerful enough to express every process that can be modelled with recursive definitions? The answer is positive, thanks to the following encoding.

A definition $A[\vec{x}] := Q$ is translated to a process $!(c_A(\vec{x}).\widehat{Q})$ where c_A is a channel associated with $A \in PID$ and fresh. $\widehat{Q}$ is obtained from Q by replacing every process instance $A[\vec{a}]$ with the process $\overline{c}_A\langle\vec{a}\rangle$, for any $A \in PID$. Now, given a P, Δ program in π-calculus, the π-calc+repl. process $\widehat{P} \parallel \prod \{ !(c_A(\vec{x}).Q) \mid A[\vec{x}] := Q \in \Delta \}$ is an equivalent system not using definitions but using replication. The symbol $\prod X$ is for parallel composition of all the processes in X.

Note that making a reduction from $A[\vec{a}] \to Q'$ requires an extra reaction

$$\overline{c}_A\langle\vec{a}\rangle \parallel !(c_A(\vec{x}).Q) \to Q[\vec{a}/\vec{x}] \parallel !(c_A(\vec{x}).Q) \to Q' \parallel !(c_A(\vec{x}).Q)$$

In fact, some presentations of the π-calculus with definitions, define the behaviour of process instances by the rule $A[\vec{a}] \to Q[\vec{a}/\vec{x}]$ when $A[\vec{x}] := Q \in \Delta$ in which case the reactions of the definition based processes coincide with the ones in their replication encoding.

A notable feature of the encoding is that all the replicated components have the shape $!(\pi.P)$ which is called "guarded replication" (π being the guard). In fact guarded replication can be easily encoded by definitions

$$!(\pi.P) \rightsquigarrow A_{!\pi.P}[fn(\pi.P)] \quad \text{with definition} \quad A_{!\pi.P}[\vec{x}] := \pi.(P \parallel A[\vec{x}])$$

Labelled Transition Semantics for π-calculus

The LTS semantics of π-calculus is more involved than the one for CCS because of mobility: it is not enough to know on which name an action is performed, we also need to know the arguments of the action and to keep track of the identity of restricted names. We thus introduce the set of actions in Act. We will, for simplicity, only consider the MONADIC case.

$Act \ni \alpha$	NAME	$subj(\alpha)$	$obj(\alpha)$	$fn(\alpha)$	$bn(\alpha)$	$\alpha\sigma$
$\overline{x}\,y$	free output	x	y	$\{x,y\}$	$\emptyset$	$\overline{\sigma(x)}\,\sigma(y)$
$x\,y$	input	x	y	$\{x,y\}$	$\emptyset$	$\sigma(x)\,\sigma(y)$
$\overline{x}\,(z)$	bound output	x	z	$\{x\}$	$\{z\}$	$\overline{\sigma(x)}\,(z)$
τ	internal	—	—	$\emptyset$	$\emptyset$	τ

$names(\alpha) := fn(\alpha) \cup bn(\alpha)$

Transition Rules

$$\text{OUT}: \frac{}{\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}y} P} \qquad \text{INP}: \frac{}{x(z).P \xrightarrow{xy} P[y/z]} \qquad \text{TAU}: \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\text{SUM}_L: \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \text{PAR}_L: \frac{P \xrightarrow{\alpha} P'}{P \| Q \xrightarrow{\alpha} P' \| Q} \; bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{COMM}_L: \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'} \qquad \text{RES}: \frac{P \xrightarrow{\alpha} P'}{\nu z.P \xrightarrow{\alpha} \nu z.P'} \; z \notin names(\alpha)$$

$$\text{OPEN}: \frac{P \xrightarrow{\overline{x}z} P'}{\nu z.P \xrightarrow{\overline{x}(z)} P'} \; z \neq x \qquad \text{CLOSE}_L: \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \| Q \xrightarrow{\tau} \nu z.(P' \| Q')} \; z \notin fn(Q)$$

$$\text{DEF}: \frac{P[\vec{z}/\vec{x}] \xrightarrow{\alpha} P'}{A[\vec{z}] \xrightarrow{\alpha} P'} \; (A[\vec{x}] := P) \in \Delta$$

Omitted are the rules

$\text{SUM}_R, \text{PAR}_R, \text{COMM}_R, \text{CLOSE}_R$

which are symmetric versions of the $-_L$ rules where the order of the operands of $\|$ or $+$ is exchanged.

Example To see how mobility is handled by the transition rules, we show how a reaction from the server/clients example can be derived.

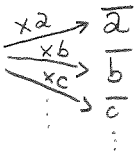
① Note the application of the substitution $[^m/_x]$ where m is the subject of the action $s\,m$

② Note the way CLOSE and OPEN cooperate to "implement" the necessary scope extrusion that makes m flow to $A[x,d]$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{s}\langle m\rangle \xrightarrow{\overline{s}m} \mathbf{0}}\,\text{OUT}}{Q[s,m] \xrightarrow{\overline{s}m} \mathbf{0}}\,\text{DEF}}{Q[s,m] \| C'[s,m] \xrightarrow{\overline{s}m} \mathbf{0} \| C'[s,m]}\,\text{PAR}_L}{\nu m.\,(Q[s,m] \| C'[s,m]) \xrightarrow{\overline{s}(m)} \mathbf{0} \| C'[s,m]}\,\text{OPEN}\ ②$$

$$\text{②}\ \text{CLOSE}_R\ \dfrac{\text{DEF}\ \dfrac{\text{①}\ \text{INP}\ \dfrac{}{s(x).(\nu d.A[x,d] \| S[s]) \xrightarrow{sm} \nu d.A[m,d] \| S[s]}}{S[s] \xrightarrow{sm} \nu d.A[m,d] \| S[s]} \qquad \nu m.(Q[s,m] \| C'[s,m]) \xrightarrow{\overline{s}(m)} \mathbf{0} \| C'[s,m]}{S[s] \| \nu m.(Q[s,m] \| C'[s,m]) \xrightarrow{\tau} \nu m.(\nu d.A[m,d] \| S[s] \| \mathbf{0} \| C'[s,m])}$$

Properties of the LTS semantics

- The LTS of π-calculus processes is NOT finitely branching: for example $x(z).\overline{z}$ 

  but it has the weaker property of being image-finite up to α-conversion.
  Image finiteness requires that for each $\alpha \in Act$ the relation $\xrightarrow{\alpha}$ is finitely branching.
  Image finiteness is enough to guarantee the stratification characterisation of bisimulation (see notes on Bisimulation)

- If $P \xrightarrow{\alpha} P'$ then $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$
  provided that if $\alpha = \overline{x}(z)$ then $z \notin fn(P\sigma) \cup range(\sigma)$

- The reaction relation $\longrightarrow$ and $\xrightarrow{\tau}$ coincide up to $\equiv$
  $P \longrightarrow P'$ iff $\exists P'' : P \xrightarrow{\tau} P'' \equiv P'$

Bisimulation for π-calculus

Now that we have a LTS semantics we can study the induced notion of strong bisimulation.

Some basic result carry over from CCS:

Theorem $\equiv$ is a strong bisimulation

In the π-calculus we have a surprise that hinders the development of algebraic bisimilarity proof techniques:

Theorem Strong bisimilarity $\sim$ is preserved by all elementary contexts apart from $x(\vec{y}).[\,]$ (the input context). Therefore $\sim$ is not a congruence.

The reader check that $\sim$ is preserved by the non-input contexts by adapting the proof for the analogous result on CCS.

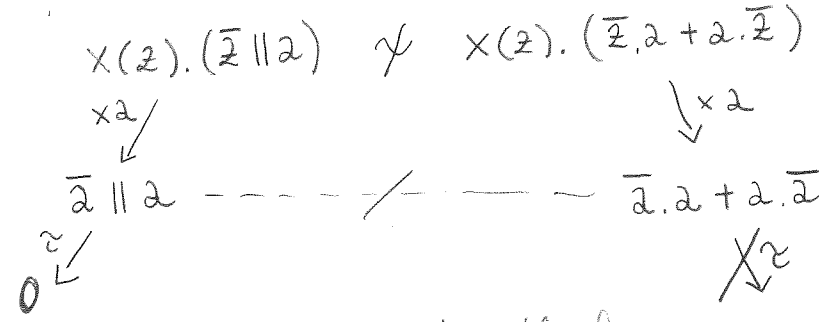
Let us see what makes this fail for input contexts.

Consider the (CCS) processes $\overline{z} \,|\!|\, a$ and $\overline{z}.a + a.\overline{z}$.

As we learned from CCS we have

$$\overline{z} \,|\!|\, a \sim \overline{z}.a + a.\overline{z}$$

but when plugged in the context $x(z).[\,]$ we get



The problem arises when z is bound to the free name a: then the first process has an internal reaction that the second doesn't have.

In fact, to be able to be indistinguishable under an input context, two processes P and Q do not just have to be bisimilar but they need to be bisimilar no matter which substitution gets applied to them, i.e. If $P\{y/z\} \sim Q\{y/z\}$ for every $y \in fn(P) \cup fn(Q) \cup \{z\}$ then $x(z).P \sim x(z).Q$.

This motivates the following definition:

Def Strong Full Bisimilarity is the relation $\simeq$ where $P \simeq Q$ if $P\sigma \sim Q\sigma$ for every substitution σ.

Theorem
- $P \simeq Q \Rightarrow P \sim Q$
- $P \equiv Q \Rightarrow P \simeq Q$

Theorem Strong Full Bisimilarity is a process congruence

In our counterexample, we now have $\overline{z} \| a \not\simeq \overline{z}.a + a.z$ because of $\sigma = [a/z]$: $\overline{a} \| a \not\sim \overline{a}.a + a.\overline{a}$.

Similar results apply to the weak form of bisimulation.

Automatic Verification for the π-calculus

We now turn our attention to the reaction semantics and try to answer the question: WHAT CAN BE DECIDED ABOUT THE INTERNAL BEHAVIOUR OF A π-CALCULUS PROGRAM?

First, let us motivate the problem by considering a concrete example. Consider again the server/client example. We study a slightly more complicated version where there are multiple servers, each with their clients. We model this scenario by modifying the environment's definition to

$$E[s] := \tau.(\nu m.\, C[s,m] \parallel E[s]) + \tau.(\nu s'.(S[s'] \parallel E[s']) \parallel E[s])$$

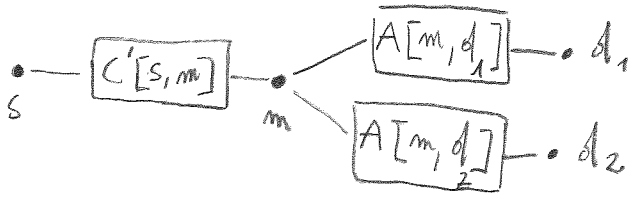
where the second τ action can spawn a new independent pair of server/environment on a fresh server address s'.
Overall, the system $\nu s.(S[s] \parallel E[s])$ can reach configurations with an arbitrary number of servers, each with an arbitrary number of connected clients.

An example of a safety property we might want to check is: can any client have more than one answer pending on its mailbox? We refer to this property as a capacity bound of 1 on the mailboxes. Let us give this property a formal meaning. We are asking if from $P_0 = \nu s.(S[s] \parallel E[s])$ we can reach any process Q where a client has at least 2 answers $A[m,d]$ pending:

$$\exists Q:\ P_0 \longrightarrow^* Q \ \equiv\ \nu m.\nu d_1.\nu d_2.(A[m,d_1] \parallel A[m,d_2] \parallel Q')$$

for some Q'. Note the use of $\equiv$: we are not looking for a violation for a specific mailbox in a specific configuration, thanks to $\equiv$ we are looking for any mailbox violating the property.

In terms of communication topologies, we are asking if we can reach a topology that contains the subgraph



This motivates the following definition.

Def The TERM EMBEDDING relation $\sqsubseteq$ is defined by

$$P \sqsubseteq P' \quad \text{if} \quad P \equiv \nu\vec{x}.Q \text{ and } P' \equiv \nu\vec{x}.(Q \parallel R)$$

Proposition $P \sqsubseteq P'$ if and only if $std(P) = \nu\vec{x}.Q$ and $std(P') = \nu\vec{x}.\nu\vec{y}.(Q \parallel R)$. $fn(Q) \cap \vec{y} = \emptyset$

Proof ($\Leftarrow$) is immediate by scope extrusion of $\vec{y}$.

($\Rightarrow$) is similarly easy: from $P \sqsubseteq P'$ we have $P \equiv \nu\vec{x_1}.Q_1$ and $P' \equiv \nu\vec{x_1}.(Q_1 \parallel R_1)$; now let $std(Q_1) = \nu x_2.Q_2$ and $std(R_1) \equiv \nu y_2.R_2$ we have $std(P) = \nu x_1 \nu x_2.Q_2$, $std(P') = \nu x_1 \nu x_2 \nu y_2.(Q_2 \parallel R_2)$.

Remark Term embedding coincides with the 'induced subgraph embedding' on communication topologies $\sqsubseteq_G$. That is: $P \sqsubseteq Q$ iff $G[\![P]\!] \sqsubseteq_G G[\![Q]\!]$. We only present this informally: we do not use this fact in proofs but it helps intuition. (up to α-conversion of P and Q)

Two graphs are in the embedding relation $G_1 \sqsubseteq_G G_2$ if there is an injective map φ from the nodes of G_1 to the ones of G_2 such that the labels are preserved and the edges are preserved: $\forall n \in nodes(G_1) : \big(\lambda_1(n) = \lambda_2(\varphi(n))$ and

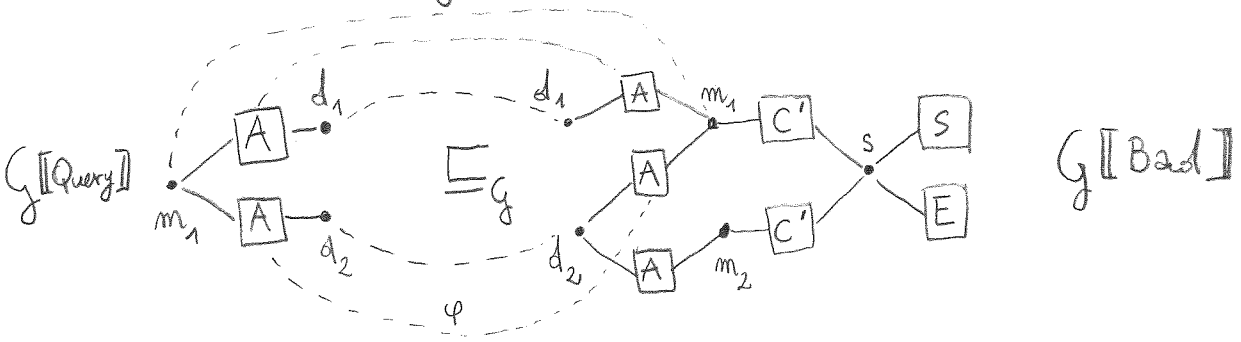
$$\forall n' \in nodes(G_2) : (n,n') \in E_1 \Leftrightarrow (\varphi(n), \varphi(n')) \in E_2\big)$$

For bipartite graphs $nodes(S, N, E, \lambda) = S \uplus N$ and φ maps nodes of S_1 to S_2 and N_1 to N_2. For comm. top. this is implied by preservation of labels.

Example Let $Query = \nu m.(\nu d.A[m,d] \parallel \nu d\, A[m,d])$

$$Bad = \nu s.\Big(S[s] \parallel E[s] \parallel \nu m_1.\big(C'[s,m_1] \parallel \nu d_1.A[m_1,d_1] \parallel \nu m_2.(C'[s,m_2] \parallel \nu d_2.(A[m_1,d_2] \parallel A[m_2,d_2]))_{\neq}\big)_{\neq}\Big)$$

We have $Query \sqsubseteq Bad$ and, correspondingly:



Note:
- We omitted the arguments in the process labels to uncluttter the picture. They can be inferred from the terms.
- We applied α-conversion to Query in order to get a comm. top. that could be directly embedded into the one of Bad
- The restricted names of Query can be reused by other processes in Bad, for example this happens for d_2. This shows the difference between term embedding and just putting a term in a parallel context.
- The dashed lines show a map φ from the nodes of $G[[Query]]$ to the ones of $G[[Bad]]$ that witnesses the $\sqsubseteq_G$ relation between the graphs.

Another example: $P = \nu x\, y.(A[x,a] \parallel B[y,x])$ we have

$$P \sqsubseteq \nu x\, y\, z.(A[z,a] \parallel B[x,y] \parallel B[y,z])$$
$$P \not\sqsubseteq \nu x\, y.(A[x,b] \parallel B[y,x])$$
$$P \not\sqsubseteq \nu x\, y.(A[x,a] \parallel B[x,y])$$
$$P \not\sqsubseteq \nu x\, y.(A[x,a] \parallel C[y,x])$$

Given a class of π-calculus processes $\mathcal{C}$ and a finite set of definitions Δ (usually left implicit) we associate to it the QOTS $(\mathcal{C}/\equiv, \rightarrow/\equiv, \sqsubseteq/\equiv)$.

The fact that this is a well-defined concept is a consequence of the following.

Proposition 1) $\sqsubseteq$ is a quasi-order

2) If $P \equiv P'$ and $Q \equiv Q'$, $P \sqsubseteq Q$ iff $P' \sqsubseteq Q'$

Proof left as an easy exercise

Since we also know, by definition, reactions are closed under $\equiv$ we usually omit the quotients $/\equiv$ and work with processes P understanding that we are really manipulating their class $[P]_\equiv$.

Proposition 1) $\rightarrow/\equiv$ is finitely branching

2) $\rightarrow/\equiv$ is decidable

3) $\sqsubseteq/\equiv$ is decidable

Proof again an easy exercise

Now that we have a QOTS, we can study two important verification problems for π-calculus.

REACHABILITY: Given P, Q determine if $P \rightarrow^* Q$

COVERABILITY: Given P, Q determine if $\exists Q'$: $P \rightarrow^* Q'$ and $Q \sqsubseteq Q'$

Note that COVERABILITY is often much more useful than reachability, which requires the specification of the full target configuration, while commonly we just consider targets any configuration that contains an "error" pattern.

In the notes on CCS we studied the implementation of counters that can be incremented (inc), decremented if not zero (dec) and tested for zero (zero). The implementation C_0 used finitely many definitions and is thus also a valid π-calculus implementation. As it is well known, two counter machines (2CM) are Turing powerful and have thus an undecidable halting problem. It is an easy exercise to encode a counter machine M into a CCS process P_M using the names $\vec{c}_i = inc_i, dec_i, zero_i$ to control counter $i = 0, 1$. We thus have:

Theorem COVERABILITY and REACHABILITY are undecidable for CCS processes, and thus for π-calculus processes.

Proof Given a 2CM M we can construct a CCS process P_M that terminates iff M terminates. Wlog we can assume the machine M terminates by executing the instruction at location h, which is encoded by $H[\vec{c}_0, \vec{c}_1]$ in P_M. We therefore have that M halts iff $H[\vec{c}_0, \vec{c}_1]$ is coverable from P_M. Reachability is similarly shown undecidable by assuming M brings all counters to 0 before terminating and using coverability in the same way. □

In the EXERCISE SHEET 11, Problem 2 we defined ν-free CCS which is the fragment where restrictions are not part of the syntax. We have shown in the sheet that REACHABILITY and COVERABILITY are both decidable for ν-free CCS by encoding processes to Petri nets. A similar result holds for ν-free π-calculus.

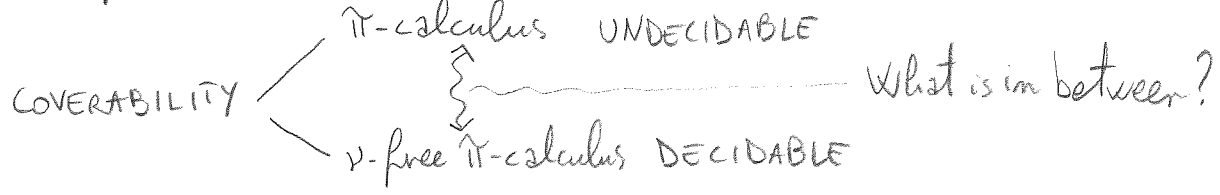
Theorem REACHABILITY and COVERABILITY are decidable for ν-free π-calculus.

Proof Assume P, Δ are normalised. It is easy to see that since restrictions are disallowed P and every reachable configuration will have the form, for some $n \in \mathbb{N}$

$$A_1[\vec{a}_1] \parallel \dots \parallel A_n[\vec{a}_n] \qquad \text{with } A_i \in P_{\mid\Delta} \ \vec{a}_i \subseteq fn(P)$$

Since the definitions Δ are finite, there are only finitely many process identifiers $P_{\mid\Delta_\Delta}$ that may appear in a reachable configuration so the set $S = \{A[\vec{a}] \mid A \in P_{\mid\Delta_\Delta}, \vec{a} \subseteq fn(P)\}$ is finite. We can then construct a Petri net $(S, T, W) = N_\Delta$ where the transitions are dictated by the definitions in the obvious way. Every reachable configuration Q is isomorphic to a marking M_Q mapping each place $A[\vec{a}]$ to the number of its occurrences in the parallel composition.
Reachability and Coverability from P then reduced to the corresponding problems on the net N_Δ from M_P, which are decidable. □

Let us focus on coverability. We have:



The motivation of the rest of the lecture is finding a fragment of π-calculus that is 1) more powerful than Petri nets
2) has a decidable coverability problem.
As hinted by the above results, such fragment has to forbid the use of restrictions that allows for the implementation of counters with zero-test.

Before we define a suitable fragment, let us analyse the reasons for the undecidability of cover. in the π-calculus.

From the results on WSTS we know that coverability is decidable for them provided they are effective and minpre is computable. So, we saw before the LTS of π-calculus is effective; therefore there needs to be at least one point from the following that fails:

- $\equiv$ is a wqo
- $\equiv$ is a simulation wrt reactions
- minpre is decidable

} at least one of these fail otherwise we would have an effective WSTS + minpre and coverab. would be decidable for π-calculus, a contradiction.

Which one is it that fails then?

We can actually show that $\equiv$ is a simulation and minpre is computable. To do so let us introduce a characterisation of reactions. As usual, we consider normalised programs.

Theorem NORMALISED REACTIONS: For any P, Q we have

$P \longrightarrow Q$ if and only if, either

(a) $P \equiv \nu \vec{x}.\,(A[\vec{a}] \parallel B[\vec{b}] \parallel C)$
(b) $A[\vec{a}] \triangleq x(\vec{z}).\,R + M_1$
(c) $B[\vec{b}] \triangleq \overline{x}\langle \vec{c} \rangle.\,S + M_2$
(d) $Q \equiv \nu \vec{x}.\,(R[\vec{c}/\vec{z}] \parallel S \parallel C)$

OR

(a') $P \equiv \nu \vec{x}.\,(A[\vec{a}] \parallel C)$
(b') $A[\vec{a}] \triangleq \tau.\,R + M$
(c') $Q \equiv \nu \vec{x}.\,(R \parallel C)$

Proof is a routine induction on the depth of inference and we omit it.

The theorem just states that when $P \to Q$ then P must contain two reacting components (or just one component doing a reaction) and Q must contain the result of their interaction.

Using the normalised reactions theorem we can easily show that $\sqsubseteq$ is a simulation.

Theorem Term embedding is a simulation.

Proof Assume $P_1 \longrightarrow P_2$ and $P_1 \sqsubseteq Q_1$. We need to find Q_2 such that $Q_1 \longrightarrow Q_2$ and $P_2 \sqsubseteq Q_2$.

From norm. reaction we know that since $P_1 \longrightarrow P_2$ we have

$$P_1 \equiv \nu \vec{x}.\,(A[\vec{a}] \parallel B[\vec{b}] \parallel C) \quad \text{with } A[\vec{a}] \triangleq \chi(\vec{z}).\,R + M_1$$
$$B[\vec{b}] \triangleq \overline{\chi}\langle\vec{c}\rangle.\,S + M_2$$
$$P_2 \equiv \nu \vec{x}.\,(R[\vec{c}/\vec{z}] \parallel S \parallel C)$$

assuming $P_1 \longrightarrow P_2$ was not due to a τ (the τ case is easier and similar)

From $P_1 \sqsubseteq Q_1$ we get that $Q_1 \equiv \nu \vec{x}.\,(A[\vec{a}] \parallel B[\vec{b}] \parallel C \parallel C')$.

So, again by norm. react. we have that $Q_1 \longrightarrow Q_2$ with

$$Q_2 \equiv \nu \vec{x}\,(R[\vec{c}/\vec{z}] \parallel S \parallel C \parallel C')$$

which also proves $P_2 \sqsubseteq Q_2$.

In other words, this means that, adding components in parallel to two components that can react, will never disable the reaction.

Note that the theorem does <u>not</u> imply that $\sqsubseteq$ is a simulation for *every* class of processes: if a class is not closed under reaction, we could have the situation where P_1, P_2, Q_1 all belong to the class but the Q_2 provided by the theorem may lay outside of it!

Proposition for every class of processes $\mathcal{C}$ that is closed under reaction, term embedding is a simulation over $\mathcal{C}$.

Now we establish decidability of minpre for π-calculus processes. We give the definition now to prove a conceptual point but it will be reused later. Assume we are only interested in procs with free names $\vec{f}$

Def *minpre for π-calculus*: remember that minpre is a function that satisfies $\text{minpre}(Q)\uparrow = \min(\text{pre}(Q\uparrow)\uparrow)$.

For a π-calculus process Q, we define

$$\text{minpre}(Q) := \min_{\sqsubseteq}(\mathcal{B})$$

where the finite set of processes $\mathcal{B}$ is computed as follows.

Let $\text{std}(Q) = \nu \vec{x}.\ (A_1[\vec{a}_1] \parallel \cdots \parallel A_n[\vec{a}_n])$.

For all $B_1, B_2 \in P_{ID_\Delta}$ (= the finite set of defined process ids in Δ) do:

Let $B_1[\vec{y}_1] := \sum_{i \in I_1} \pi_i.P_i$ and $B_2[\vec{y}_2] := \sum_{j \in I_2} \pi'_j.P'_j$ be the two definitions of B_1 and B_2 in Δ,

Let $\vec{b}$ be fresh names with $|\vec{b}| = |\vec{y}_1| + |\vec{y}_2|$.

For all substitutions σ_1, σ_2 with $\sigma_i : \vec{y}_i \to \vec{b} \cup \vec{x} \cup \vec{f}$ $i = 1,2$ (the globally free names, see remark below)

For all $i \in I_1$, $j \in I_2$ such that $\pi_i\sigma_1 = c(\vec{z})$ and $\pi'_j\sigma_2 = \bar{c}\langle\vec{d}\rangle$

Let $\text{std}(P_i\sigma_1[\vec{d}/\vec{z}] \parallel P_j\sigma_2) = \nu \vec{x}'.\ Q'$

Compute the smallest parallel composition of process instances C such that

$$Q \sqsubseteq \nu\vec{x}.\nu\vec{b}.\nu\vec{x}'.(Q' \parallel C)$$

(note that there are finitely many candidates for C)

If there is such C then

Add $\nu\vec{x}.\nu\vec{b}.(B_1[\vec{y}_1]\sigma_1 \parallel B_2[\vec{y}_2]\sigma_2 \parallel C)$ to $\mathcal{B}$

Remark Since if $P \to P'$ then $fn(P') \subseteq fn(P)$, when we check coverability from some P_0 it is sufficient to restrict the attention to processes with free names included in $fn(P_0)$. This is the reason why minpre is parametric in the free names $\vec{f}$: they can be set to $fn(P_0)$ when using it to answer coverability from some P_0.

So now we have that term embedding is a simulation and minpre is decidable for π-calculus terms. This means that Term embedding is *not* a wqo on π-calculus processes or we would have decidability of coverability.

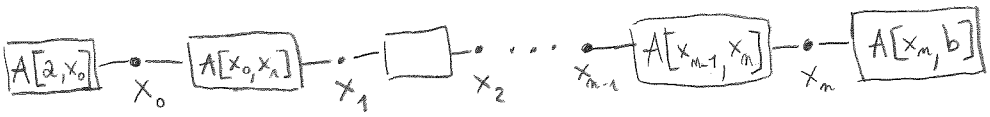
The following bad sequence is indeed a proof that $\sqsubseteq$ is not a wqo.

Let

$$Chain_n = \nu x_0 \nu x_1 \dots \nu x_n . (A[a, x_0] \parallel A[x_0, x_1] \parallel A[x_1, x_2] \parallel \cdots \parallel A[x_n, b])$$

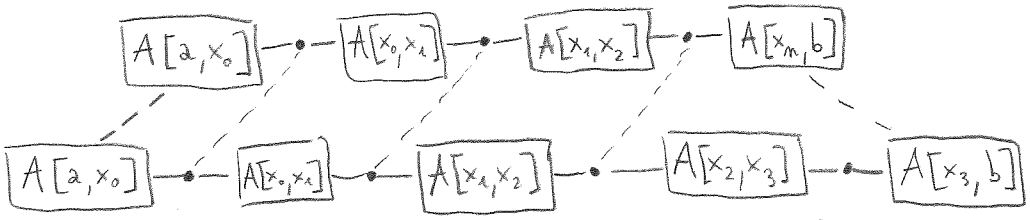
with free names $\{a, b\}$, for any $n \in \mathbb{N}$.

The comm. top. of such term has the form



The reader can check that the infinite sequence $Chain_0, Chain_1, Chain_2 \dots$ is indeed a bad sequence for $\sqsubseteq$.

Intuitively, consider the two comm. top. for $Chain_2$ and $Chain_3$



In trying to map the nodes of the former to nodes of the latter, we are free to α-convert by renaming $x_0 \dots x_2$ to be able to match them to any of the $x_0 \dots x_3$ but we cannot rename the free names a and b. Therefore the only nodes that can be mapped to from the two ends of the chain are the corresponding ends of the longer chain, as indicated by the dashed lines. Any candidate mapping would then fail to connect at least one pair of nodes that are mappings of connected nodes in the shorter chain. Any mapping would then fail to show the embedding and we have $Chain_i \not\sqsubseteq Chain_j$ for all i, j

It is interesting to note that the counter implementation C_0 exploits exactly the same kind of unbounded chaining of $Chain_n$ to represent the number n.

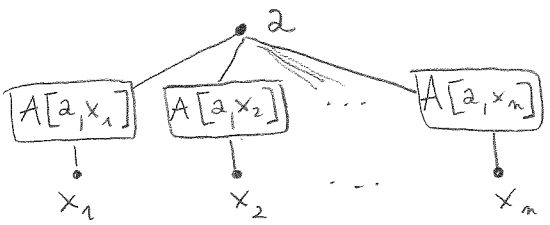
In EXERCISESHEET 13, Problem 2 we see how to implement Resettable counters, i.e. counters that can be incremented, decremented when non zero and reset to zero but not tested for zero.

In a minimalistic implementation the kind of structure used to represent a number n resembles the one of the processes $Bunch_n$ defined as follows:

$$Bunch_n := \nu a.\, \nu x_1 \ldots \nu x_n.\, (A[a,x_1] \,\|\, A[a,x_2] \,\|\, \ldots \,\|\, A[a,x_n])$$

which defines the comm. top.:

$\mathcal{G}[\![Bunch_n]\!]$ =



The definitions that follow are a way to capture the qualitative difference between the unboundedness in $Chain_n$ which we call "in depth" and the one in $Bunch_n$ which we call "in breadth".

Def NESTING of RESTRICTIONS: The function $nest_\nu$ maps processes to naturals as follows:

$$nest_\nu(\mathbf{0}) := nest_\nu(A[\vec{a}]) := nest_\nu\left(\sum_{i\in I} \pi_i.P_i\right) := 0$$

$$nest_\nu(\nu x.P) := 1 + nest_\nu(P)$$

$$nest_\nu(P \,\|\, Q) := \max(nest_\nu(P), nest_\nu(Q))$$

Def DEPTH: $depth(P) := \min\{nest_\nu(Q) \mid Q \equiv P\}$

The nest_ν function counts the maximum number of nested restrictions in a term. This measure is however too dependent on the specific way we are presenting the term:

$$\mathrm{nest}_\nu(\mathrm{Chain}_n) = n+1 \quad \text{and} \quad \mathrm{nest}_\nu(\mathrm{Bunch}_n) = n+1$$

which shows that nest_ν is not really able to distinguish the two kinds of growth. The function depth is the one that measure the intrinsically needed nesting of restrictions of a term: a nesting is not really needed if it can be removed using $\equiv$. Indeed we have for every $n \in \mathbb{N}$

$$\mathrm{depth}(\mathrm{Bunch}_n) = 2 \quad \text{as witnessed by}$$

$$\mathrm{Bunch}'_n = \nu a.(\nu x.\, A[a,x])^n \quad \text{where we write } P^n = \underbrace{P \parallel \cdots \parallel P}_{n \text{ times}}$$

which has $\mathrm{nest}_\nu(\mathrm{Bunch}'_n) = 2$ and $\mathrm{Bunch}_n \equiv \mathrm{Bunch}'_n$.
It is a bit less trivial to see that

$$\mathrm{depth}(\mathrm{Chain}_n) = \lceil \log_2 n \rceil$$

which can be obtained by restricting the middle of the chain first and proceed recursively with the left and right portions as we demonstrate for Chain_6

$$\mathrm{Chain}'_6 = \nu x_3.\Big\{\nu x_4.\big\{\nu x_0.(A[a,x_0] \parallel A[x_0,x_1]) \parallel \nu x_2.(A[x_1,x_2] \parallel A[x_2,x_3])\big\} \parallel \nu x_5.\big\{\nu x_4.(A[x_3,x_4] \parallel A[x_4,x_5]) \parallel \nu x_6.(A[x_5,x_6] \parallel A[x_6,b])\big\}\Big\}$$

which has $\mathrm{nest}_\nu(\mathrm{Chain}'_6) = 3 = \lceil \log_2 6 \rceil$ and $\mathrm{Chain}_6 \equiv \mathrm{Chain}'_6$.

So now depth is a measure that can differentiate the two kinds of unboundedness: the unboundedness of $\mathrm{Chain}_1, \mathrm{Chain}_2, \mathrm{Chain}_3 \ldots$ is reflected in the unboundedness of their depth, while the unboundedness of $\mathrm{Bunch}_1, \mathrm{Bunch}_2 \ldots$ is harmless, as indicated by a depth ≤ 2 for all n.

Remark We give here a characterisation of depth on communication topologies. We do not prove it but we report it because it is useful to help intuition.

For a bipartite graph (S, N, E, λ) a path of length n is a sequence $s_0 e_1 s_1 e_2 s_2 \cdots e_n s_n$ where $s_i \in S$ and $e_i \in N$ and $(s_i, e_{i+1}) \in E$ and $(s_{i+1}, e_{i+1}) \in E \quad \forall i \in \{0, \dots, n\}$.

A path is called SIMPLE if it does not repeat nodes from N.

Let $lsp(G)$ be the longest simple path in the bipartite graph G. Then we have

$$depth(P) \leq lsp(G[\![P]\!]) \leq 2^{depth(P)} - 1$$

Which implies that imposing a bound on the depth imposes a bound on the lsp and viceversa.

Now we are ready to define the fragment of π-calculus that we were after.

Def • Let $k \in \mathbb{N}$ and $X \subseteq \mathcal{N}$ be a finite set of names.

We define $\mathbb{D}_k^X := \{ P \mid depth(P) \leq k, fn(P) \subseteq X \}$,

to be the set of processes with depth at most k and free names in X.

- Define the set $\mathcal{R}_\Delta(P) := \{ Q \mid P \longrightarrow^* Q \}$ to be the reachable processes from P under definitions Δ.
- We say a process P is k-bounded if $\mathcal{R}_\Delta(P) \subseteq \mathbb{D}_k^{fn(P)}$
- A process is depth-bounded if it is k-bounded for some $k \in \mathbb{N}$.

Observe that $P \longrightarrow P'$ implies $fn(P') \subseteq fn(P)$ so the condition on the free names imposed by $\mathcal{R}_\Delta(P) \subseteq \mathbb{D}_k^{fn(P)}$ is trivially satisfied.

Theorem $(\mathbb{D}_k^X, \sqsubseteq)$ is a wqo for every $k \in \mathbb{N}$ and finite $X \subseteq \mathcal{N}$

Proof Outline We outline the proof first and then we proceed by proving each claim.

We define a map $\mathcal{F}[\![-]\!]$ from processes to labelled forests. We write $\mathbb{F}_L$ for the set of all finite forests labelled with elements of L and by $\sqsubseteq_{\mathbb{F}_L}$ we denote the forest embedding order (see below for the formal definitions).

We show that, for some finite set L:

① For each $P \in \mathbb{D}_k^X$, $\exists\, Q \equiv P$ such that $\mathcal{F}[\![Q]\!] \in \mathbb{F}_L$

② If $\mathcal{F}[\![Q_1]\!] \sqsubseteq_{\mathbb{F}_L} \mathcal{F}[\![Q_2]\!]$ then $Q_1 \sqsubseteq Q_2$

Let us see how ① + ② prove the theorem.
Assume, towards a contradiction, that $\mathbb{D}_k^X$ has a bad sequence $P_1, P_2, P_3 \ldots$ then by ① we would have a corresponding sequence $Q_1, Q_2, \ldots$ with $Q_i \equiv P_i$. Since $\sqsubseteq$ is invariant under $\equiv$, the sequence $Q_1, Q_2 \ldots$ is also bad. But then, by ② the sequence $\mathcal{F}[\![Q_1]\!], \mathcal{F}[\![Q_2]\!], \ldots$ would be a bad sequence of $(\mathbb{F}_L, \sqsubseteq_{\mathbb{F}_L})$ that by Kruskal's theorem is a wqo for L finite ↯

In the rest of the note we introduce the notation and lemmas necessary to prove ① and ② and complete the proof.

Let us introduce some notation for forests and recall Kruskal's theorem (for the proof see the notes on wqos).

$\mathbb{F}_L$ is the set of forests with nodes labelled by elements of L.

A forest is a finite set of disjoint trees. The general statement of Kruskal's theorem states that if the set of labels L is a wqo then the set of L-labelled trees forms a wqo. From the notes on wqos we also know that the finite subsets of a wqo form a wqo, which implies that if L is a wqo, the L-labelled forests form a wqo too. We will only need the special case where L is finite, and thus well quasi ordered by $=$.

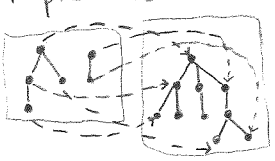
Let us recall the wqo on forests with finite labels L.

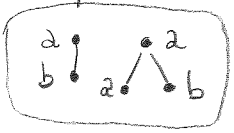
The wqo $\sqsubseteq_{\mathbb{F}_L}$ (forest embedding) is defined as follows.

For $F_1, F_2 \in \mathbb{F}_L$, we have $F_1 \sqsubseteq_{\mathbb{F}_L} F_2$ if there exist an injective function $\varphi : \text{nodes}(F_1) \to \text{nodes}(F_2)$ so that for all nodes $n \in \text{nodes}(F_1)$:

i) $\lambda_1(n) = \lambda_2(\varphi(n))$ where λ_1, λ_2 are the labelling functions of F_1, F_2 resp.
[φ preserves labels]

ii) $\forall n'$ ancestor of n in F_1 : $\varphi(n')$ is ancestor of $\varphi(n)$ in F_2
[φ preserves ancestry]

Remark
- Kruskal's theorem actually holds for the stronger order which also requires iii) nodes belonging to disjoint subtrees of F_1 are mapped to nodes of disjoint subtrees of F_2 by φ. We do not need this requirement for our result.
- Note that ii) requires φ to preserve ancestry not parenthood so a valid φ could be  provided the labels match

To make it easier to write forests we introduce the notation $\ell[F]$ to denote the tree with root labelled by $\ell \in L$ and as children the roots of the trees in the forest $F \in \mathbb{F}_L$.
So $a[\varnothing]$ is a leaf labelled with $a \in L$ and $\{a[\{b[\varnothing]\}], a[\{a[\varnothing], b[\varnothing]\}]\}$ is the forest 

.

We are now ready to define the map $\mathcal{F}[\![-]\!]$

Def FOREST ENCODING

$$\mathcal{F}[\![P]\!] := \begin{cases} \varnothing & \text{if } P = \mathbf{0} \\ \{P[\varnothing]\} & \text{if } P \text{ is sequential} \\ \{a[\mathcal{F}[\![Q]\!]]\} & \text{if } P = \nu a.\, Q \\ \mathcal{F}[\![P_1]\!] \uplus \mathcal{F}[\![P_2]\!] & \text{if } P = P_1 \parallel P_2 \end{cases}$$

Example

$$\mathcal{F}[\![Bunch'_n]\!] = \left\{ \begin{array}{c} a \\ \big| \\ x \quad x \quad \cdots \quad x \\ | \quad | \qquad | \\ A[a,x] \; A[a,x] \cdots A[a,x] \end{array} \right\} =_\alpha \left\{ \begin{array}{c} a \\ \big| \\ x_0 \quad x_1 \quad \cdots \quad x_n \\ | \quad | \qquad | \\ A[a,x_0] \; A[a,x_1] \; A[a,x_n] \end{array} \right\}$$

Obviously α-conversion on the term will induce a similar transformation on forest encodings.

$$\mathcal{F}[\![Bunch_n]\!] = \left\{ \begin{array}{c} a \\ | \\ x_0 \\ | \\ x_1 \\ \vdots \\ x_n \\ \| \\ A[a,x_0] \cdots A[a,x_n] \end{array} \right\}$$

$$\mathcal{F}[\![Chain'_6]\!] = \left\{ \begin{array}{c} \qquad\qquad x_3 \\ \qquad\quad / \quad \backslash \\ \qquad x_1 \qquad\quad x_4 \\ \qquad / \quad \backslash \qquad / \quad \backslash \\ x_0 \qquad x_2 \qquad x_5 \qquad x_6 \\ | \qquad | \qquad\quad / \quad \backslash \\ A[a,x_0] \; A[x_0,x_1] \cdots A[x_5,x_6] \; A[x_6,b] \end{array} \right\}$$

To show (1) we prove the following Lemma

Lemma Every P can be α-converted to a process P' such that $|bn(P')| = nest_\nu(P)$

Proof By induction on the structure of P

CASE P is $\mathbf{0}$ or a sequential process then $|bn(P)| = 0 = nest_\nu(P)$.

CASE $P = \nu a.\, P_1$. By induction hypothesis $\exists P_1' =_\alpha P_1$ so that $|bn(P_1')| = nest_\nu(P_1)$. Then we can always ensure $a \notin bn(P_1')$ so we have

$$|bn(\nu a.\, P_1')| = 1 + |bn(P_1')| = 1 + nest_\nu(P_1) = nest_\nu(P)$$

CASE $P = P_1 \parallel P_2$ then by ind. hyp. for $i = 1, 2$ we get $P_i' =_\alpha P_i$ with $|bn(P_i')| = nest_\nu(P_i)$ and wlog we can ensure $bn(P_1') \subseteq bn(P_2')$ or $bn(P_2') \supseteq bn(P_1')$ by renaming names. So by letting $P' = P_1' \parallel P_2' =_\alpha P$ we have

$$\begin{aligned} |bn(P')| &= |bn(P_1') \cup bn(P_2')| = \\ &= \max\left(|bn(P_1')|, |bn(P_2')|\right) = \\ &= \max\left(nest_\nu(P_1), nest_\nu(P_2)\right) = nest_\nu(P) \end{aligned}$$

□

Example $P = \nu a.\, (\nu x_0.\, A[a, x_0] \parallel \cdots \parallel \nu x_n.\, A[a, x_n]) =_\alpha Bunch_n'$

and although $nest_\nu(P) = nest_\nu(Bunch_n') = 2$ we have $|bn(P)| = n+1$ while $|bn(Bunch_n')| = 2$.

Proof of (1) Let $Y \subseteq \mathcal{N}$ be such that $Y \cap X = \emptyset$ and $|Y| = k$.

Let $L = \{A[\vec{a}] \mid \vec{a} \subseteq X \cup Y, |\vec{a}| = |\vec{x}|, A[\vec{x}] := Q \in \Delta\} \cup Y$.

L is finite because Δ is finite. For any $P \in \mathbb{D}_k^X$ we have, by definition of depth that $\exists Q \equiv P$ with $nest_\nu(Q) = depth(P) \leq k$. By the lemma above, we can assume such Q has $|bn(Q)| = nest_\nu(Q) \leq k$. So we can take it to have $bn(Q) \subseteq Y$.

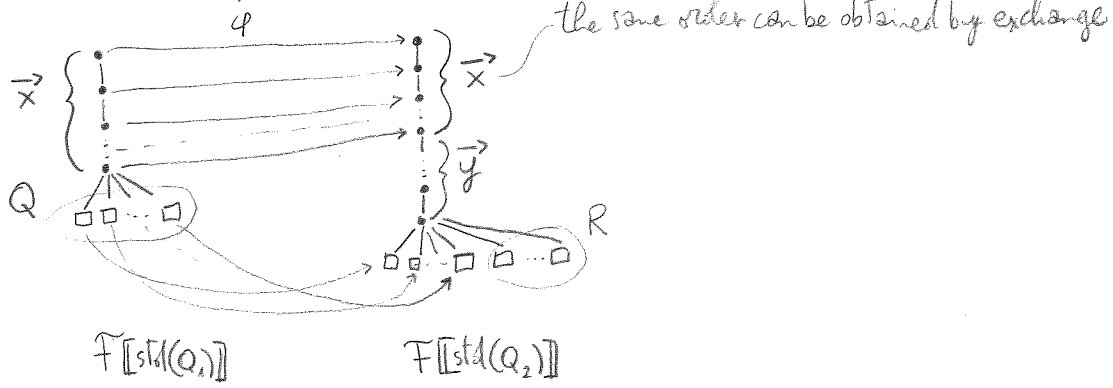
Then, by definition of $\mathcal{F}[\![-]\!]$ we have $\mathcal{F}[\![Q]\!] \in \mathbb{F}_L$ for L as above. □

Proof of ② Let Q_1, Q_2 be s.t. $\mathcal{F}[\![Q_1]\!] \subseteq_{\mathcal{FL}} \mathcal{F}[\![Q_2]\!]$

which means that we have an injective label and ancestry preserving function φ from the nodes of $\mathcal{F}[\![Q_1]\!]$ to the ones of $\mathcal{F}[\![Q_2]\!]$.

To show that $Q_1 \sqsubseteq Q_2$, we first α-convert the terms, by α-converting the two forests so that 1) every restricted name is distinct and 2) φ is still label preserving after the α-conversion.

After this disambiguation step we can use scope extrusion to bring Q_1 and Q_2 to their normal form. The forest encodings of the two standard forms will look like



which shows $Q_1 \equiv \nu\vec{x}.Q$ and $Q_2 \equiv \nu\vec{x}\,\nu\vec{y}\,(Q \parallel R)$ proving $Q_1 \sqsubseteq Q_2$.

The correspondence between $\vec{x}$ and $\vec{Q}$ in the two terms is ensured by φ. □

This concludes the proof that $(\mathbb{D}_k^{\times}, \sqsubseteq)$ is a wqo.

Theorem Let P be a k-bounded process, for $k \in \mathbb{N}$.

Then $(R_\Delta(P)_{/\equiv}, \rightarrow_{/\equiv}, \sqsubseteq_{/\equiv})$ is a WSTS

Proof Direct consequence of $R_\Delta(P) \subseteq \mathbb{D}_k^{\#(P)}$ which is a wqo and the fact that $R_\Delta(P)$ is closed under reduction by def. which makes the proof of simulation for $\sqsubseteq$ applicable. □

Theorem Termination is decidable for depth-bounded processes.

Proof By instantiating the Finite Reachability Tree algorithm on the WSTS $(R_\Delta(P)_{/\equiv}, \rightarrow_{/\equiv}, \sqsubseteq_{/\equiv})$: the FRT applies because as we argued $\rightarrow_{/\equiv}$ is finitely branching and computable and $\sqsubseteq_{/\equiv}$ is computable. The fact that $R_\Delta(P)_{/\equiv}$ is a wqo ensures termination of the algorithm.

Theorem Coverability is decidable for k-bounded processes.

Proof By instantiating the Backward Search algorithm on the WSTS $(R_\Delta(P_0)_{/\equiv}, \rightarrow_{/\equiv}, \sqsubseteq_{/\equiv})$: as we have seen minpre is computable but if we apply the algorithm literally, we would need to be able to compute $\text{minpre}(Q) \cap R_\Delta(P_0)$ which would seem to require solving the problem of reachability. We cannot simply use minpre because it can produce processes of arbitrary depth. The problem is easily fixed by observing that computing $\text{minpre}(Q) \cap \mathbb{D}_k^{f_m(P_0)}$ ensures that the fixpoint in the BWD search will include $R_\Delta(P_0)$ iff the target can be covered from P_0. Restricting to $\mathbb{D}_k^{f_m(P_0)}$ will at the same time ensure termination by wqo. Observe that for this reason it is crucial here to know the value of k, while in the termination algorithm, the existence of a k was sufficient.

While for the BWD search we need to know K, coverability can be decided even when it is not known in advance.
To prove this we instantiate the forward search which requires that we provide a symbolic representation for ideals. We give the main construction and omit the proofs.
Ideals for depth-bounded processes can be represented by expressions following the grammar

$$L ::= \mathbf{0} \mid A[\vec{a}] \mid L \parallel L \mid \nu a.\, L \mid L^\omega$$

Again we assume we are only dealing with normalised processes

Most definitions extend naturally from processes to ideals by adding the case for L^ω. For $\equiv$ we do not add laws for L^ω (we could add $L^\omega \equiv L \parallel L^\omega$, $(L^\omega)^\omega = L^\omega$, $\mathbf{0}^\omega \equiv \mathbf{0}$ but they are not necessary for the theory).
For nest_x we set $\mathrm{nest}_x(L^\omega) = \mathrm{nest}_x(L)$ and $\mathrm{depth}(L) = \min\{\mathrm{nest}_x(L') \mid L' \equiv L\}$ is now well defined. These definitions are justified by the following mapping from expressions L to the ideal they represent: $[\![L]\!]$:

$$[\![\mathbf{0}]\!] := \{\mathbf{0}\} \qquad [\![A[\vec{a}]]\!] := \{A[\vec{a}], \mathbf{0}\} \qquad [\![\nu a.\, L]\!] := \{\nu a.\, P \mid P \in [\![L]\!]\}_{/\equiv}$$

$$[\![L_1 \parallel L_2]\!] := \{P_1 \parallel P_2 \mid P_1 \in [\![L_1]\!],\ P_2 \in [\![L_2]\!]\}_{/\equiv}$$

$$[\![L^\omega]\!] := \{P_1 \parallel \dots \parallel P_n \mid P_i \in [\![L]\!],\ n \in \mathbb{N}\}_{/\equiv}$$

Using this symbolic representation, one can show that

- the set of expressions L with $\mathrm{depth}(L) \le K$ and $\mathrm{fn}(L) \subseteq X$ represent all and only the ideals of the wqo $(\mathbb{D}^X_{K/\equiv}, \sqsubseteq_{/\equiv})$. [If $P \in [\![L]\!]$ $\mathrm{depth}(P) \le \mathrm{depth}(L)$]
- $\widehat{\mathrm{post}}$ is computable: it only requires "expanding" the ω a bounded num. of times.
- $[\![L_1]\!] \subseteq [\![L_2]\!]$ is decidable

Theorem Coverability is decidable for depth-bounded processes

Proof By instantiating the Forwards Search algorithm on the WSTS $(\mathcal{R}_{\Delta}(P_0)_{/\equiv}, \rightarrow_{/\equiv}, \sqsubseteq_{/\equiv})$ which, as we mentioned, has a post-effective ideal completion. The algorithm does not need to know for which k P_0 is k-bounded: this is because by enumerating finite unions of expressions $L_1 \dots L_n$ all downward closed sets of depth at most k are considered, for all $k \in \mathbb{N}$. For the positive coverability instances, the algorithm would reply 'YES' because the forward reachable config. exploration will eventually find a covering path. For the negative instances, the alg. will reply 'NO' because the dwdcl. set enumeration will eventually consider the symbolic repr. of the dwdcl set $\mathcal{R}(P_0)\downarrow$ which is a fwd. ind. invariant containing P_0 but not the target, showing that the target is not coverable. Note that overapproximations of $\mathcal{R}(P_0)\downarrow$ could be enough to determine the same result.

Theorem Given P and $k \in \mathbb{N}$ it is decidable to determine if P is k-bounded.

Proof By adapting the FwB Search. We run two semialgorithms in parallel:

(A) We do a fwd reach. exploration stopping with 'NO' if we reach a process with depth exceeding k. This will eventually happen *iff* P is not k-bounded

(B) We enumerate all finite unions of expressions $L_1 \dots L_n$ with $\text{depth}(L_i) \leq k$ and check if any is the repr. of a fwd. ind. invariant containing P. Such invariant will be eventually found *iff* P is k-bounded.

Example The servers/clients example can be shown depth bounded by checking that the following is a forward ind. invariant containing $\nu s.(S[s] \| E[s])$:

$$L = \left(\nu s.\left(S[s] \| E[s] \| \left(\nu m.\, C[s,m]\right)^{\omega} \| \left(\nu m.\left(C'[s,m] \| Q[s,m]\right)\right)^{\omega} \| \left(\nu m.\left(C'[s,m] \| \nu d.\, A[m,d]\right)\right)^{\omega}\right)\right)^{\omega}$$

Since L does not contain Query, L is a proof that no mailbox will have more than one pending answer.

Before showing some other application of these results, let us present two "surprises". The first is that we cannot use depth bounds for bounded model checking. In bounded model checking one defines a measure μ on configurations and asks whether an error configuration can be reached when only configurations c with $\mu(c) \leq K$ for some fixed K. A simple instance could be checking reachability of a config of a Counter Machine restricting to configurations where the value of the counters does not exceed a fixed K (here $\mu(q, c_0, c_1) = \max(c_0, c_1)$). In this example the problem becomes decidable since there are only finitely many config. if the counters are bounded. Of course this is quite trivial but the idea can be used to do more sophisticated analysis thanks to more sophisticated measures μ). Here one might ask if using $\mu = \text{depth}$ one gets decidability of the bounded model checking problem. This cannot be derived from the results we presented because $(\mathbb{D}_K^X, \to, \sqsubseteq)$ is NOT a WSTS : while $\sqsubseteq$ is a wqo, it is not a simulation on $\mathbb{D}_K^X$. The reader can check this fact by finding a counterexample where $P_1, P_2, Q_1 \in \mathbb{D}_K^X$ $P_1 \to P_2$ and $P_1 \sqsubseteq Q_1$ but the Q_2 with $Q_1 \to Q_2 \sqsupseteq P_2$ (which exists) is <u>not</u> in $\mathbb{D}_K^X$.

In fact one can prove that depth bounded model checking is not decidable:

Theorem The following problem is undecidable:

Given $P, Q, k \in \mathbb{N}$ determine if there exist $P_1 \dots P_n$ with $P \to P_1 \to \dots \to P_n \sqsupseteq Q$ and $\text{depth}(P_i) \leq k$.

Proof idea By reducing the halting problem. The idea of the reduction is that with a depth bounded process we can weakly simulate a 2CM. The honest runs will stay with depth $\leq k$. The cheating runs are so that as soon as one cheats the depth increases above k.

Another natural question is wether reachability is decidable for k-bounded processes.

Theorem Reachability is undecidable for k-bounded processes for all $k \geq 1$. (For $k=0$ we get the ν-free π-calculus)

Proof As we have seen in EXERCISE SHEET 13, Problem 2, it is possible to implement resettable counters with 1-bounded processes. We can therefore replicate the proof of undecidability of reachability for Reset Nets. We encode a 2CM M with a process P_M with two resettable counters. When we guess one of the counters is zero (which we cannot check directly) we also reset it. At this point we can recognise the wrong guesses by checking if there is any "garbage" left behind by a reset of a counter that was not zero. Since the reach. target can specify "no garbage" we reduced reachability for M to reachability of the 1-bounded P_M. □

Theorem All terminating processes are depth-bounded.

Proof A terminating process is one from which no infinite reaction sequences start. Since $\rightarrow_{/\equiv}$ is finitely branching we have that for terminating processes the set $\mathcal{R}_{\Delta}(P)_{/\equiv}$ is finite and thus depth-bounded. □

From this simple fact we derive the second surprise:

Theorem Depth-boundedness is undecidable.

Proof Towards a contradiction, assume that we can decide if a given P is depth-bounded. As we already noted, for every CM M we can write a CCS process P_M that terminates iff M halts. Now we can decide the halting problem by checking if P_M is depth-bounded:

- if it is depth-bounded we can decide if it terminates using the FRT algorithm
- if it is <u>not</u> depth-bounded it is <u>not</u> terminating by the above theorem. ↯ □

Thus we have that while checking k-boundedness for a specific $k \in \mathbb{N}$ is decidable, although non primitive recursive, checking if such k exists is undecidable. However coverability is decidable even if k is not known.

We conclude with an application to expressivity of replication and then a summary of the fragments we considered.

Let CCS! be the CCS calculus where no definitions are allowed but with the addition of the bang operator !P. The same operation, as we have discussed, does not alter the expressivity in the case of π-calculus but there the encoding was relying on mobility to mimic the substitution mechanism offered by definitions. The question is then: is CCS! as expressive as CCS?

Theorem Every process in CCS! is depth bounded.

Proof See exercise sheet 13. The idea is that the function δ

$$\delta(\mathbf{0}) = 0 \qquad \delta\left(\sum_{i\in I} \alpha_i.P_i\right) = \max\{\delta(P_i) \mid i \in I\}$$

$$\delta(\nu a.P) = 1 + \delta(P) \qquad \delta(P_1 \| P_2) = \max(\delta(P_1), \delta(P_2))$$

$$\delta(!P) = \delta(P)$$

bounds the depth of each reachable term:

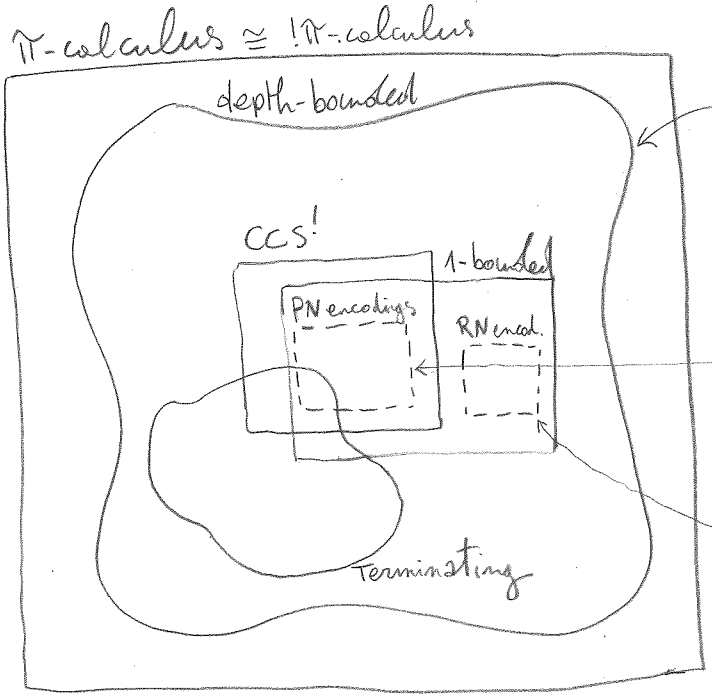
if $P \to Q$ then $\delta(P) \geq \delta(Q')$ for some $Q' \equiv Q$

and $\delta(P) \geq \text{depth}(P)$.

This answers our question: since coverability is undecidable for CCS but decidable for CCS! we know that replication in CCS is strictly less expressive than definitions.

(It has actually been proven that reachability for CCS! is decidable!)

Overall we get the following picture:



The curvy boundary indicates that membership to the fragment is undecidable

decidable coverability

decidable reachability

undecidable reachability

↑ undecidable coverability (Turing powerful)