Process algebras

Representing concurrent systems

- LTSs have a strong mathematical foundation (needed to apply formal methods)

- But they are harder to manipulate directly (e.g., via drawing graphs) the bigger they become
  - Imagine having to code a large software project using flowcharts instead of programming languages…

- Process algebras = formal languages for structured textual description of concurrent systems

Process algebras: common elements

- A process is made of elementary actions

- Smaller processes can be composed to create larger ones, by means of specific operators

- Typically, those operators need to describe:
  - Sequencing (a system does a, then b, then …)
  - Choice (a system may do a, or b, or …)
  - Parallel composition (a system does a and b in parallel)

CCS: Actions

CCS = Calculus of communicating systems

Intuitively, given a set of channel names *A*:

- CCS processes can perform input/output actions on that channel
  - Input: a, output: $\bar{a}$, where a ∈ *A*
  - We say that a, $\bar{a}$ are complementary
- They may synchronise on complementary actions
- They can perform an invisible action (denoted τ)
  - Synchronisation on τ is not allowed

CCS: Processes

- Grammar:

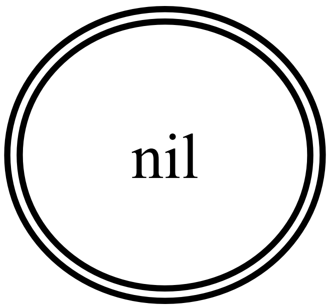
P, Q ::= nil	(idle process)
| μ.P	(action prefix) [μ is an action]
| P + Q	(choice)
| P | Q	(parallel composition)
| P \ a	(restriction) [a is a visible action]
| P [a/b]	(relabelling) [a,b are actions]
| K	(named process invocation)

Structural operational semantics

- A CCS process is just a term (a piece of text)
- We must give a rigorous meaning to every term
- One possible approach: operational semantics
  - Define an LTS for every CCS term
  - Each state in the LTS is a CCS term
  - States are linked by labelled transitions
  - The set of transitions is defined via inference rules
- If rules are based on the syntax of the language, we have a structural operational semantics (SOS)

CCS: Idle process

- The idle process **nil** (or **0**) cannot do anything
- Its LTS is a single state with no transitions
- Thus, there are no SOS rules associated to **nil**

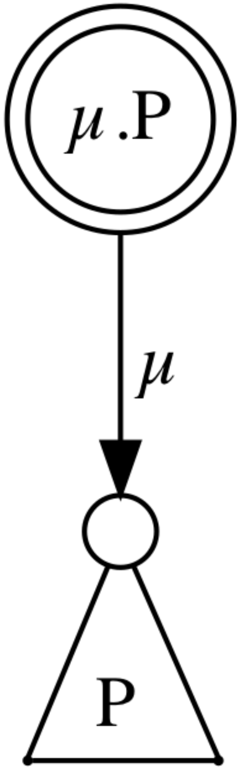


CCS: action prefix

- μ.P performs μ and continues as P
- μ can be either:
  - A channel name a
  - A co-name $\bar{a}$
  - The invisible action τ
- We will assume that $\bar{\bar{a}} = a$
- Semantics of μ.P:

$$\frac{}{\mu . P \xrightarrow{\mu} P}$$

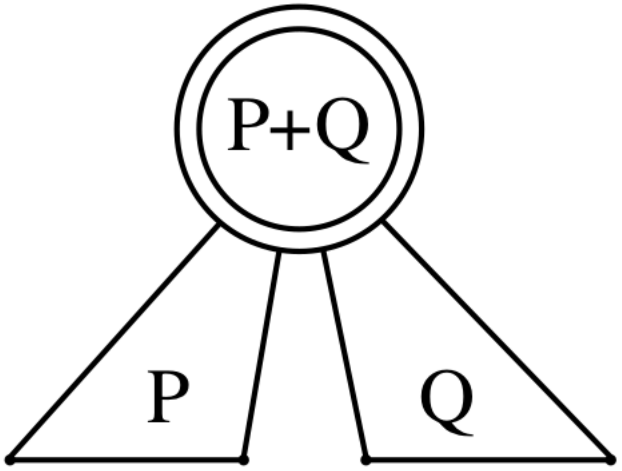
- No premises = this rule always holds



CCS: choice

- P + Q behaves either as P or as Q
- If P can perform an action and become P', then P+Q may also do that (same for Q, Q')

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \frac{Q \xrightarrow{\mu} Q'}{P + Q \xrightarrow{\mu} Q'}$$



CCS: Parallel composition

- P | Q executes P and Q in parallel
- Furthermore, if P can perform an action named a and Q can perform its complement ā, then a rendezvous may happen
- The result is an invisible action τ

  (= only binary rendezvous)

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{Q \xrightarrow{\mu} Q'}{P \mid Q \xrightarrow{\mu} P \mid Q'} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Exercise

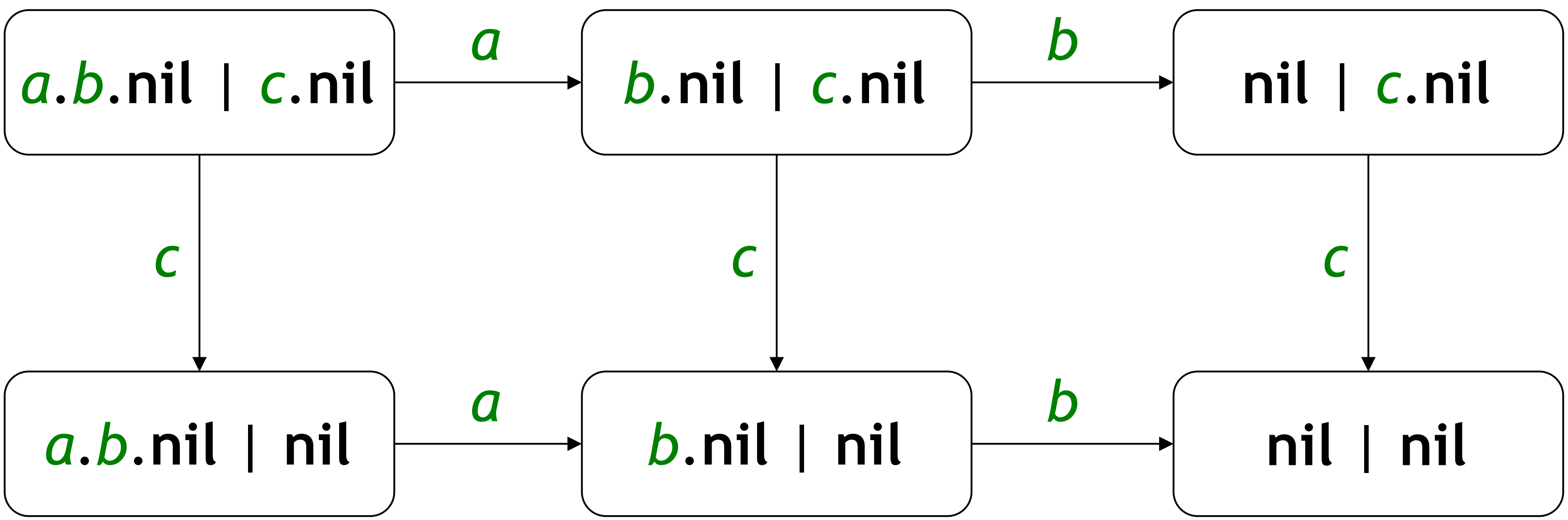
Draw the LTS corresponding to the CCS term

$$(a.b.\mathbf{nil} \mid c.\mathbf{nil})$$

Solution

Draw the LTS corresponding to the CCS term

$$(a.b.\mathbf{nil} \mid c.\mathbf{nil})$$



Exercise

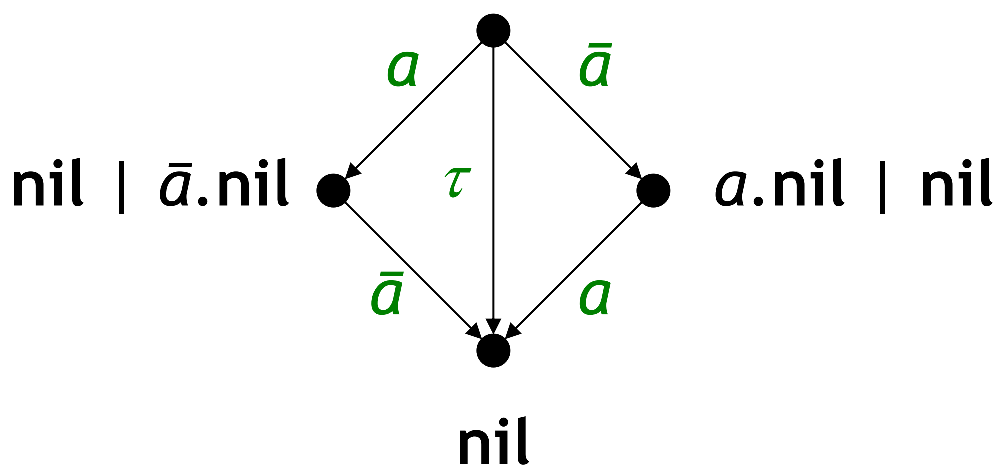
Draw the LTS corresponding to the CCS term

$$(a.\mathbf{nil} \mid \bar{a}.\mathbf{nil})$$

Solution

Draw the LTS corresponding to the CCS term

$$(a.\mathbf{nil} \mid \bar{a}.\mathbf{nil})$$



CCS: restriction

- P \ a can perform the same transitions as P, except those labelled a (or $\bar{a}$)
- Useful to force synchronisation:
  - (a.nil | $\bar{a}$.nil) can perform a, $\bar{a}$, and τ
  - (a.nil | $\bar{a}$.nil) \ a can only perform τ
  - τ cannot be restricted
- P \ {a, b, c, …} is the same as P \ a \ b \ c \ …

$$\frac{P \xrightarrow{\mu} P' \quad \mu \neq a \qquad \mu \neq \bar{a}}{P \setminus a \xrightarrow{\mu} P' \setminus a}$$

CCS: relabelling (1/2)

- P [a/b] behaves exacly like P, except that it performs a (or $\bar{a}$) whenever P would do b (or $\bar{b}$)
  - Actions can be relabelled to τ (hiding)
  - τ cannot be relabelled
  - You cannot relabel a and $\bar{a}$ to different actions
- Multiple relabellings: P [a/b, c/d, …]
- a/b actually represents a relabelling function, i.e., a function from actions to actions that satisfies the description above

CCS: relabelling (2/2)

- Properties of a relabelling function f:
  - $f(\tau) = \tau$ (the internal action is not renamed)
  - $f(\bar{x}) = \overline{f(x)}$ for all visible actions (co-name relations are preserved)
- a/b is the function f such that
  - $f(b) = a$, $f(\bar{b}) = \bar{a}$
  - $f(x) = x$ for all other actions x

$$\frac{P \xrightarrow{\mu} P'}{P[f] \xrightarrow{f(\mu)} P'[f]}$$

CCS: named process invocation

- A named process is a CCS term P that is given a name K. We write $K \triangleq P$, "K is defined as P"
- CCS terms can contain names: they are equivalent to their definitions
  - E.g. if $K \triangleq c.\mathrm{nil}$, then $a.b.K = a.b.c.\mathrm{nil}$
- This allows recursion e.g., $K \triangleq a.b.K$
  - $K = a.b.a.b.a.b. \ldots$

$$\frac{P \xrightarrow{\mu} P' \quad K \triangleq P}{K \xrightarrow{\mu} P'}$$

CCS: conclusions

- The above rules are enough to formally describe the behaviour of any CCS term
- With this formal semantics, we can prove that two processes are bisimilar (equivalence checking)
  - http://caal.cs.aau.dk/ (CAAL: online automated tool)

Other process algebras

- Value-passing CCS
- CSP (Communicating Sequential Processes)
- ACP (Algebra of Communicating Processes)
- LOTOS (Language of Temporal Ordering Specifications)
- LNT (LOTOS New Technology), etc.

- They introduce operators and constructs that make it easier to specify complex systems