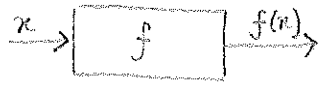
INTRODUCTION TO CONCURRENCY

- beginning of computer science (1930's)



  - a program computes a function
  - non-termination must be avoided (although it cannot be removed).
  - plenty of computational models (Turing machines, λ-calculus, ...) — all equivalent
- the present: cloud computing, wireless sensor networks, ...
  - "programs" are compositions of interacting processes (possibly non-terminating)
  - correctness ≠ "computes the given function", as in sequential programs
  - correctness = reliability (wrt. Murphy's law) + security (wrt. attackers)
  - writing correct concurrent software is hard

    $x := 1;\ x := x+1$ equivalent to $x := 2$? (in sequential models they denote the same f)

  If we take a context $\bullet \mid x := 2$, by replacing • with the above:

    $x := 1; x := x+1 \mid x := 2$ may evaluate to 3

    $x := 2 \mid x := 2$ always evaluate to 2

  This has two consequences
  - processes are not functions (evaluation depends on the context)
  - (sequential) equivalence is not a congruence (no compositional semantics)
- Why do we need a "theory" of concurrency?
  - "The price for reliability is the pursuit of the utmost simplicity. It is a price which the very rich find most hard to pay" — Tony Hoare, 1980
  - "Almost anything in software can be implemented, sold, and even used given enough determination." (T.H. 1980)
  - Evolution of sequential programming languages:
    - ASM, FORTRAN, COBOL: poor structures, no guarantees
    - Java, Haskell, Ocaml: rich structure (abstractions), guarantees (TYPE SAFETY)
    - Why Haskell/Ocaml have all these nice properties?
      - build upon the λ-calculus, a minimal model for sequential programs
      - all extensions involve formal methods to show that they preserve the properties of the basic model
        - for instance, integers are not needed in the pure λ-calculus (Church numerals), but they can be added while preserving type safety.

Towards a basic calculus for concurrency. Design principles:

- algebraic theory: define atomic processes, to combine through operators
- crucial choice: how to define interaction between processes.
  - one-to-one, one-to-many, many-to-one, many-to-many (which is the most primitive?)
  - to interact, two processes must share a NAME (e.g. a channel name)
  - interaction = synchronous communication

$$\overline{n}\langle 3\rangle \mid n(y).\overline{z}\langle y+1\rangle \longrightarrow \overline{z}\langle 4\rangle \quad \left(= \overline{z}\langle y+1\rangle\{^{3}/_{y}\}\right)$$

  - notation:
    - $\overline{n}\langle 3\rangle$ denotes output of 3 on channel n
    - $n(y)$ denotes input on channel n. y is a "formal parameter"
    - $.$ in $n(y).\overline{z}\langle y+1\rangle$ denotes sequential composition
    - $\mid$ denotes parallel composition

- example (memory cell). Write a process P such that:
  - if U does $\overline{put}\langle v\rangle$, then U "writes" v in P
  - if U does $get(n)$, then U "reads" the value written in P

$$P \triangleq put(n).P'\langle n\rangle \qquad P'(n) \triangleq \overline{get}\langle n\rangle.P$$

$$U \triangleq \overline{put}\langle 5\rangle.get(y).U'$$

$$U \mid P \rightarrow get(y).U' \mid P'\langle 5\rangle \rightarrow U'\{^{5}/_{y}\} \mid P$$

  - exercise: how do you obtain a memory cell that can interact correctly with $U \triangleq \overline{put}\langle 5\rangle.\overline{put}\langle 3\rangle.get(y).U'$?

$$P \triangleq put(n).P'\langle n\rangle \qquad P'(n) = \overline{get}\langle n\rangle.P + put(z).P'\langle z\rangle$$

$$U \mid P \rightarrow \overline{put}\langle 3\rangle.get(y).U' \mid \overline{get}\langle 3\rangle.P + put(z).P'\langle z\rangle$$
$$\rightarrow get(y).U' \mid P'\langle 3\rangle \qquad \left(= \overline{get}\langle 3\rangle.P + put(z).P'\langle z\rangle\right)$$
$$\rightarrow U'\{^{3}/_{y}\} \mid P$$

  - notation: $+$ denotes non-deterministic choice
  - note that P is a non-terminating process
  - exercise: make $U = \overline{put}\langle 5\rangle.get(n).get(y).U' \mid P$ non-stuck

$$P = put(n).P'\langle n\rangle \qquad P'(n) = \overline{get}\langle n\rangle.P'(n) + put(z).P'\langle z\rangle$$

- Communication vs. synchronization

  Sometimes the actual values passed are immaterial, e.g. in

  $\overline{x}\langle v\rangle.P \mid x(y).Q$ with y not occurring (free) in Q.

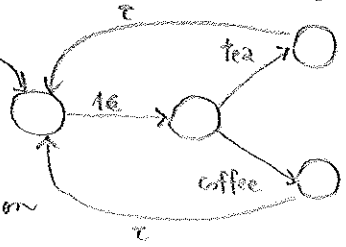
  In such cases we abbreviate: $\overline{x}.P \mid x.Q \longrightarrow P \mid Q$

  Now the fact that $\overline{x}$ is an output and x is an input is irrelevant; it suffices that x and $\overline{x}$ are complementary to have synchronization.

- example: coffee machine

  The machine is specified as an LTS:

  where τ denotes an internal action

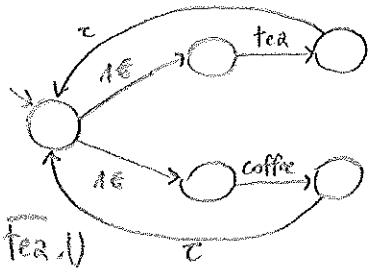


$$V \triangleq 1€.(tea.\tau.V + coffee.\tau.V)$$

$$U \triangleq \overline{1€}.\overline{tea}.U$$

$$V \mid U \longrightarrow tea.\tau.V \mid \overline{tea}.U \longrightarrow \tau.V \mid U \longrightarrow V \mid U$$

- example: a dishonest coffee machine



$$V' \triangleq 1€.tea.\tau.V' + 1€.coffee.\tau.V$$

$$V' \mid U = 1€.tea.\tau.V' + 1€.coffee.\tau.V' \mid \overline{1€}.\overline{tea}.U$$

$$\longrightarrow coffee.\tau.V' \mid \overline{tea}.U$$

- A note on observational equivalence.

  - Note that if we interpret the above transition systems as FSA, where the initial state is accepting, they both recognize the same language: $(1€ \cdot (tea + coffee) \cdot \tau)^*$

  - This contrasts with the fact that the machines V, V' are "observationally" different.

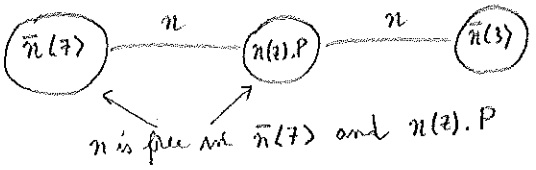
  - Process calculi require a finer notion of equivalence (wrt. FSA). Several observational theories exist, e.g.
    - bisimulation
    - barbed congruence
    - ...

- names can be communicated as "values" too

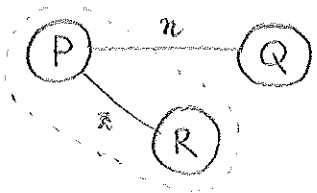
$$\bar{n}\langle y\rangle \mid n(u).\bar{u}\langle 7\rangle \longrightarrow \bar{u}\langle 7\rangle\{^{y}/_{u}\} = \bar{y}\langle 7\rangle$$

- communicating names = creating links between processes
  - global names ≈ all processes connected to each other (not very realistic)
  - local names ≈ only processes which share a name can communicate
- how to define the scope of a name ?
  - $(\nu n)\,P$ means that name n is local to P
  - $(\nu n)\,P \equiv (\nu y)\,P\{^{y}/_{n}\}$, i.e. bound names can be α-converted
- example: $((\nu n)(\bar{n}\langle 7\rangle \mid n(z).P) \mid \bar{n}\langle 3\rangle \xrightarrow{?} \dots P\{^{3}/_{z}\}$
  - no: $(\nu n)(\bar{n}\langle 7\rangle \mid n(z).P) \mid \bar{n}\langle 3\rangle \equiv (\nu y)(\bar{y}\langle 7\rangle \mid y(z).P) \mid \bar{n}\langle 3\rangle$
    $\longrightarrow ((\nu y)\,P\{^{7}/_{z}\}) \mid \bar{n}\langle 3\rangle$
  - graphical representation:

  - without the "νn" :



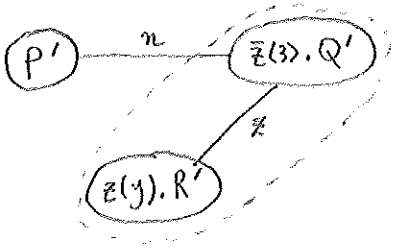
- mobility
  - $P = \bar{n}\langle z\rangle.P'$ $\quad R = z(y).R'$ $\quad Q = n(y).\bar{y}\langle 3\rangle$
  - consider the process $((\nu z)\,P|R) \mid Q$ $\quad$ z not free in P', y not free in Q'



  - after a transition: $(\nu z)(\underline{\bar{n}\langle z\rangle}.P' \mid z(y).R') \mid \underline{n(y)}.\bar{y}\langle 3\rangle.Q'$

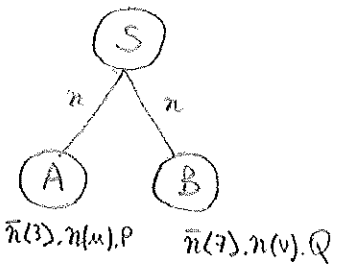
$$\longrightarrow (\nu z)(P' \mid z(y).R' \mid \bar{y}\langle 3\rangle\{^{z}/_{y}\}.Q')$$

$$\equiv P' \mid (\nu z)(z(y).R' \mid \bar{z}\langle 3\rangle.Q') \quad (z \notin fn(P'))$$



- exercise: write a server which increments the values it receives (and sends them back on the same channel)

$S \triangleq a(u).\ \bar{a}\langle u+1\rangle \mid S$



$S \mid \underline{\bar{a}\langle 3\rangle}.a(u).P \mid \underline{\bar{a}\langle 7\rangle}.a(v).Q$

$\rightarrow \bar{a}\langle 4\rangle \mid S \mid a(u).P \mid \underline{\bar{a}\langle 7\rangle}.a(v).Q$

$\rightarrow \underline{\bar{a}\langle 4\rangle} \mid \bar{a}\langle 7\rangle \mid S \mid a(u).P \mid \underline{a(v).Q}$

$\rightarrow \underline{\bar{a}\langle 7\rangle} \mid S \mid \underline{a(u).P} \mid Q\{4/v\}$

$\rightarrow S \mid P\{7/u\} \mid Q\{4/v\}$ ⚠ this is wrong

- solution: use a "private" name for the return value

$S \triangleq a(u,z).\ \bar{z}\langle u+1\rangle \mid S$ (note that S is polyadic)

$S \mid (\nu y)\ \bar{a}\langle 3,y\rangle.y(u).P \mid (\nu w)\ \bar{a}\langle 7\rangle.w(v).Q)$

$\rightarrow (\nu y)(\ \bar{y}\langle 4\rangle \mid S \mid y(u).P \mid (\nu w)\ \bar{a}\langle 7\rangle.w(v).Q\ )$

$\equiv (\nu y)(\ \bar{y}\langle 4\rangle \mid y(u).P) \mid S \mid (\nu w.\ \bar{a}\langle 7\rangle.w(v).Q)$

$\rightarrow \ldots \mid (\nu w)(\ \bar{w}\langle 8\rangle \mid S \mid w(v).Q\ )$

It is no longer possible to replay the "attack" shown above. This is because the name "y" is private to A and the first invocation of S, while "w" is private to B and the second S. With "private" we mean that there are no free occurrences of y in B, nor in the second instantiation of S.

- $(\nu a)\,P$ delimits to P the scope of a. In other words, it binds the free occurrences of a in P

- We can extend the scope of a name by communicating it:

$(\nu y)(\ \bar{a}\langle 3,y\rangle.y(u).P) \mid S \rightarrow (\nu y)(\ y(u).P \mid \bar{y}\langle 4\rangle\ ) \mid S$

- Syntax (monadic version = each message consists of one name)
  - assume a set $\mathcal{N}$ of names (infinite, ranged over $x, y, z, \ldots$)
  - processes (of the "core" π-calculus without sum)

$P, Q ::=$	0	nil process	$fn(0) = \emptyset$
	$P \mid Q$	parallel	$fn(P \mid Q) = fn(P) \cup fn(Q)$
	$(\nu x)P$	restriction	$fn((\nu x)P) = fn(P) \setminus \{x\}$
	$x(y).P$	input	$fn(x(y).P) = \{x\} \cup (fn(P) \setminus \{y\})$
	$\bar{x}\langle y \rangle.P$	output	$fn(\bar{x}\langle y \rangle.P) = \{x, y\} \cup fn(P)$
(replaces recursion)	$!P$	replication ($= P \mid P \mid \cdots$)	$fn(!P) = fn(P)$

- Structural congruence. We want to identify pairs of processes P, Q such that Q can play the role of P, in all possible contexts.
  - context: a process with a hole $\bullet$:

$$\mathcal{C} ::= \bullet \mid \bullet \mid P \mid P \mid \bullet \mid (\nu x)\bullet \mid x(y).\bullet \mid \bar{x}\langle y \rangle.\bullet \mid !\bullet$$

  - idea: define an equivalence relation $\equiv$ such that,

$$P \equiv Q \text{ implies } \mathcal{C}(P) \equiv \mathcal{C}(Q)$$

  - DEF: $\equiv$ is the smallest congruence relation such that:
    - $P \equiv Q$ if P and Q are α-convertible
    - $P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
    - $(\nu x)0 \equiv 0 \qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$

      $(\nu x)(P \mid Q) = (\nu x)P \mid Q$ if $x \notin fn(Q)$
    - $!P \equiv P \mid !P$
  - exercise: $x \notin fn(Q) \Rightarrow (\nu x)Q \equiv Q \qquad (\nu x)Q \equiv (\nu x)(Q \mid 0) \equiv (\nu x)(0 \mid Q)$
    $\equiv (\nu x)0 \mid Q \equiv 0 \mid Q \equiv Q \mid 0 \equiv Q$
  - exercise $P \equiv Q \Rightarrow fn(P) = fn(Q)$

- Reduction semantics

$$\overline{n}\langle y\rangle.P \mid n(z).Q \rightarrow P \mid Q\{y/z\} \qquad [\text{REACT}]$$

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q}\ [\text{PAR}] \qquad \frac{P \rightarrow P'}{(\nu n)P \rightarrow (\nu n)P'}\ [\text{RES}]$$

$$\frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}\ [\text{STRUCT}]$$

- Substitution ($P\{n/y\}$ substitutes n for all the free occurrences of y in P)

$$0\{n/y\} = 0 \qquad (P|Q)\{n/y\} = P\{n/y\} \mid Q\{n/y\}$$

$$((\nu z)P)\{n/y\} = \begin{cases} (\nu z)P & \text{if } z = y \\ (\nu z)P\{n/y\} & \text{otherwise} \end{cases} \qquad (!P)\{n/y\} = !(P\{n/y\})$$

$$(z(v).P)\{n/y\} = \begin{cases} z\{n/y\}(v).P & \text{if } v = y \\ z\{n/y\}(v).P\{n/y\} & \text{otherwise} \end{cases} \qquad (\overline{z}\langle v\rangle.P)\{n/y\} = \overline{z}\langle v\rangle\{n/y\}.P\{n/y\}$$

- exercise: $(n(y).P \mid R) \mid \overline{n}\langle z\rangle.Q \equiv n(y).P \mid (R \mid \overline{n}\langle z\rangle.Q)$

$$\equiv n(y).P \mid (\overline{n}\langle z\rangle.Q \mid R) \equiv (n(y).P \mid \overline{n}\langle z\rangle.Q) \mid R$$

$$\equiv (\overline{n}\langle z\rangle.Q \mid n(y).P) \mid R \rightarrow \quad \text{⚠ show derivation tree}$$

$$(Q \mid P\{z/y\}) \mid R \equiv P\{z/y\} \mid R \mid Q$$

- exercise. Let $P = (\nu y)(\overline{n}\langle y\rangle.0)$

$$n(z).\overline{a}\langle z\rangle.0 \mid !P$$

$$\equiv n(z).\overline{a}\langle z\rangle.0 \mid (\nu y)(\overline{n}\langle y\rangle.0) \mid !P$$

$$\equiv (\nu y)(n(z).\overline{a}\langle z\rangle.0 \mid \overline{n}\langle y\rangle.0) \mid !P$$

$$\rightarrow (\nu y)(\overline{a}\langle y\rangle.0 \mid 0) \mid !P$$

$$\equiv (\nu y)(\overline{a}\langle y\rangle.0) \mid !P$$

- polyadic π-calculus

$$P ::= \bar{x}\langle y_1 \dots, y_m\rangle.P \quad \text{output } (\bar{x} \text{ when } m=0)$$
$$x(y_1, \dots, y_m).P \quad \text{input } (x \text{ when } m=0)$$

$$\bar{x}\langle y_1 \dots, y_m\rangle.P \mid x(z_1 \dots, z_m).Q \longrightarrow P \mid Q\{y_1/z_1 \dots y_m/z_m\}$$

- is there an encoding from polyadic π to monadic π ?

- a naïve encoding

$$[\bar{x}\langle y_1 \dots y_m\rangle.P] = \bar{x}\langle y_1\rangle \dots \bar{x}\langle y_m\rangle.[P]$$
$$[x(y_1 \dots y_m).P] = x(y_1) \dots x(y_m).[P]$$

- example : $x(y_1 y_2).P \mid \bar{x}\langle z_1 z_2\rangle.Q \mid \bar{x}\langle z_1' z_2'\rangle.Q'$

- two possible reductions :

$$\dots \longrightarrow P\{z_1/y_1\ z_2/y_2\} \mid Q \mid \bar{x}\langle z_1' z_2'\rangle.Q' \quad , \text{ or}$$
$$\dots \longrightarrow P\{z_1'/y_1\ z_2'/y_2\} \mid \bar{x}\langle z_1 z_2\rangle.Q \mid Q'$$

- encoding :

$$x(y_1).x(y_2).P \mid \bar{x}\langle z_1\rangle.\bar{x}\langle z_2\rangle.Q \mid \bar{x}\langle z_1'\rangle.\bar{x}\langle z_2'\rangle.Q'$$
$$\longrightarrow x(y_2).P\{z_1/y_1\} \mid \bar{x}\langle z_2\rangle.Q \mid \bar{x}\langle z_1'\rangle.\bar{x}\langle z_2'\rangle.Q'$$
$$\longrightarrow P\{z_1/y_1\}\{z_1'/y_2\} \mid \bar{x}\langle z_2\rangle.Q \mid \bar{x}\langle z_1'\rangle.Q' \quad \text{⚠ No}$$

- the correct encoding (which avoids mix-ups)

$$[\bar{x}\langle y_1 \dots y_m\rangle.P] = (\nu w)\ \bar{x}\langle w\rangle.\bar{w}\langle y_1\rangle \dots \bar{w}\langle y_m\rangle.[P]$$
$$[x(y_1 \dots y_m).P] = x(v).v(y_1) \dots v(y_m).[P]$$

- example above :

$$x(v).v(y_1).v(y_2).P \mid (\nu w)(\bar{x}\langle w\rangle.\bar{w}\langle z_1\rangle.\bar{w}\langle z_2\rangle.Q) \mid (\nu w')(\bar{x}\langle w'\rangle.\bar{w}'\langle z_1'\rangle.\bar{w}'\langle z_2'\rangle.Q')$$
$$\longrightarrow (\nu w)(w(y_1).w(y_2).P \mid \bar{w}\langle z_1\rangle.\bar{w}\langle z_2\rangle.Q) \mid \dots$$
$$\longrightarrow (\nu w)(w(y_2).P\{z_1/y_1\} \mid \bar{w}\langle z_2\rangle.Q) \mid \dots$$
$$\longrightarrow (\nu w)(P\{z_1/y_1\}\{z_2/y_2\} \mid Q) \mid \dots$$

- encoding recursion into replication
  - recursive definition : $K(\vec{n}) \triangleq P$ (n may occur free in P)
  - constant application : $K\langle \vec{v} \rangle$
  - reduction rule : $$\frac{K(\vec{n}) \triangleq P}{K\langle \vec{v} \rangle \rightarrow P\{\vec{v}/\vec{n}\}}$$

- example : $P(x) \triangleq W(y).P(y) + \bar{r}\langle x \rangle.P(x)$

  memory cell containing x (W = write, R = read)

$P\langle 0 \rangle \mid \bar{W}\langle 3 \rangle . \bar{R}(n) . Q$

$\rightarrow P\langle 3 \rangle \mid R(n) . Q$

$\equiv (W(y).P(y) + \bar{r}\langle 3 \rangle . P\langle 3 \rangle) \mid R(n) . Q$

$\rightarrow P'\langle 3 \rangle \mid Q\{3/n\}$

- encoding the example :

$$[P\langle 0 \rangle] = (\nu a)(\bar{a}\langle 0 \rangle \mid !a(n).P')$$

$$P' = W(y).\bar{a}\langle y \rangle + \bar{R}\langle n \rangle . \bar{a}\langle n \rangle$$

$[P\langle 0 \rangle] \mid \bar{W}\langle 3 \rangle . R(n) . Q$

$\equiv (\nu a)(\bar{a}\langle 0 \rangle \mid a(n).P' \mid !a(n).P' \mid \bar{W}(3).R(n).Q)$

$\rightarrow (\nu a)(P'\{0/n\} \mid !a(n).P' \mid \bar{W}\langle 3 \rangle . R(n) . Q)$

$\equiv (\nu a)((W(y).\bar{a}\langle y \rangle + \bar{R}\langle 0 \rangle . \bar{a}\langle 0 \rangle) \mid !a(n).P' \mid \underline{\bar{W}\langle 3 \rangle}.R(n).Q)$

$\rightarrow (\nu a)(\bar{a}\langle 3 \rangle \mid !a(n).P' \mid R(n).Q)$

$\equiv (\nu a)(\underline{\bar{a}\langle 3 \rangle} \mid \underline{a(n)}.P' \mid !a(n).P' \mid R(n).Q)$

$\rightarrow (\nu a)(P'\{3/n\} \mid !a(n).P' \mid R(n).Q)$

$\equiv (\nu a)((W(y).\bar{a}\langle y \rangle + \underline{\bar{R}\langle 3 \rangle}.\bar{a}\langle 3 \rangle) \mid !a(n).P' \mid \underline{R(n)}.Q)$

$\rightarrow (\nu a)(\bar{a}\langle 3 \rangle \mid !a(n).P' \mid Q\{3/n\})$

- encoding booleans
  - true : $T(a) \triangleq a(x,y).\,\bar{x}$
  - false : $F(a) \triangleq a(x,y).\,\bar{y}$
  - if-then-else : $R = (\nu t)(\nu f)\left(\bar{b}\langle t,f\rangle.\,(t.P \mid f.Q)\right)$ $\quad t,f \notin fn(P|Q)$

- example :

$$R \mid T\langle b\rangle \to (\nu t)(\nu f)\left(\bar{b}\langle t,f\rangle.\,(t.P \mid f.Q) \mid b(x,y).\,\bar{x}\right)$$
$$\equiv (\nu t)(\nu f)\left(\underline{\bar{b}\langle t,f\rangle}.\,(t.P \mid f.Q) \mid \underline{b(x,y).\,\bar{x}}\right)$$
$$\to (\nu t)(\nu f)\left(\underline{t}.P \mid f.Q \mid \underline{\bar{t}}\right)$$
$$\to (\nu t)(\nu f)\left(P \mid f.Q\right)$$
$$\equiv P \mid (\nu t)(\nu f)\, f.Q$$
$$\approx P \qquad (\approx \text{ means "observationally equivalent"})$$

- exercise : $R \mid F\langle b\rangle \to^{*} Q \mid (\nu t)(\nu f)\, t.P$

- encoding integers
  - integer κ : $[\kappa](n) \triangleq\ !n(z,o).(\bar{o}^{\kappa}).\,\bar{z}$
    where : $\bar{o}^{\kappa}$ abbreviates $\bar{o}.\bar{o}.\ldots.\bar{o}$ (κ times)
  - successor : $succ(n,y) \triangleq\ !n(z,o).\,\bar{o}.\bar{y}\langle z,o\rangle$
  - test for zero : $zero(a,b) \triangleq\ \bar{a}\langle z,o\rangle.\,(z.T\langle b\rangle \mid o.F\langle b\rangle)$

- examples :
  - $[2]\langle a\rangle \mid \bar{a}\langle z,o\rangle.\,o.o.z$
    $\to\ !a(z,o).\,\bar{o}.\bar{o}.\bar{z} \mid \bar{a}\langle z,o\rangle.\,o.o.z \to \bar{o}.\bar{o}.\bar{z} \mid o.o.z \mid [2]\langle a\rangle$
  - $succ\langle a,b\rangle \mid [1]\langle b\rangle \mid \bar{a}\langle z,o\rangle.\,o.o.z$
    $\to\ !\underline{a(z,o)}.\,\bar{o}.\,\bar{b}\langle z,o\rangle \mid !b(z,o).\,\bar{o}.\bar{z} \mid \underline{\bar{a}\langle z,o\rangle}.\,o.o.z$
    $\to\ \bar{o}.\,\bar{b}\langle z,o\rangle \mid !b(z,o).\,\bar{o}.\bar{z} \mid o.o.z \mid succ\langle a,b\rangle$
    $\to\ \bar{b}\langle z,o\rangle \mid b(z,o).\,\bar{o}.\bar{z} \mid o.z \mid succ\langle a,b\rangle \mid [1]\langle b\rangle \to \bar{o}.\bar{z} \mid o,z \mid \ldots$
  - $zero\langle a,b\rangle \mid [o]\langle a\rangle \to \bar{a}\langle z,o\rangle.\,(z.T\langle b\rangle \mid o.F\langle b\rangle) \mid !a(z,o).\,\bar{z}$
    $\to\ \underline{z}.T\langle b\rangle \mid o.F\langle b\rangle \mid \underline{\bar{z}} \mid [o]\langle a\rangle \to T\langle b\rangle \mid \ldots$