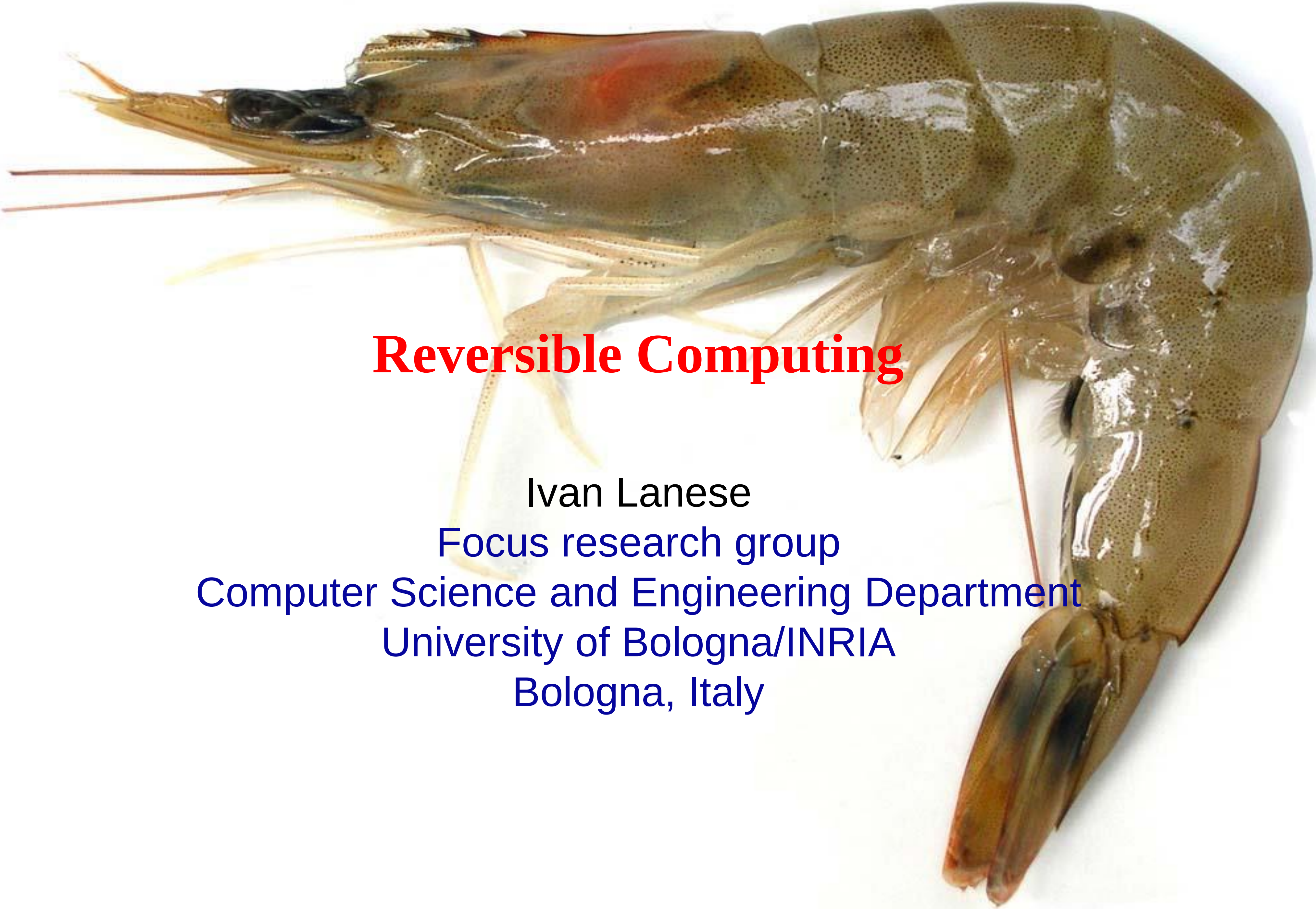


Reversible Computing

Ivan Lanese
Focus research group
Computer Science and Engineering Department
University of Bologna/INRIA
Bologna, Italy

ρCCS: reminder

- Causal-consistent Reversible CCS
- Syntax:

$$M ::= k : P \mid [a, P, P', k, k', k'', k'''] \mid k \prec k', k'' \mid$$
$$M|M' \mid \nu u\, M \mid 0$$
$$P ::= a.P \mid \bar{a}.P \mid P + P' \mid P|P' \mid \nu a\, P \mid 0$$

- Reduction rules:

$$k : (\bar{a}.P + Q)|k' : (a.P' + Q')$$
$$\rightarrow \nu h\, \nu h' \;\; h : P|h' : P'|[a, Q, Q', k, k', h, h']$$
$$h : P|h' : P'|[a, Q, Q', k, k', h, h']$$
$$\rightsquigarrow k : (\bar{a}.P + Q)|k' : (a.P' + Q')$$

Example

$$k : \bar{a}.P \mid k' : (a.b.0 + a.c.0) | k'' : \bar{b}.(Q|Q') \;\; \rightarrow$$
$$\nu h\, \nu h' [a, 0, a.c.0, k, k', h, h'] | h : P | h' : b.0 | k'' : \bar{b}.(Q|Q')$$
$$\rightarrow$$
$$\nu h\, \nu h'\, \nu l\, \nu l' [a, 0, a.c.0, k, k', h, h'] | h : P | [b, 0,0, k'', h', l', l]$$
$$| l : 0 | l' : (Q|Q') \rightsquigarrow$$
$$\nu h\, \nu h'\, \nu l\, \nu l' [a, 0, a.c.0, k, k', h, h'] | h : P | h' : b.0 | k'' : \bar{b}.(Q|Q')$$
$$\rightsquigarrow$$
$$\nu h\, \nu h'\, \nu l\, \nu l' k : \bar{a}.P \mid k' : (a.b.0 + a.c.0) | k'' : \bar{b}.(Q|Q')$$

Programming in ρCCS

- We said that the programmer will write processes as usual, only the runtime support should change
- Here we have a lot of additional information
- Where does the additional information comes from?

ρCCS vs CCS

- Given a CCS process P we can generate a ρCCS configuration as $\nu k\ k : P$
  - No memories
  - No causal dependencies
- The programmer writes the CCS process and we can transform it into a ρCCS configuration
- Given a ρCCS configuration we can generate a CCS process by removing all the additional information
- The two transformations form a Galois connection
  - α from ρCCS to CCS
  - c from CCS to ρCCS

ρCCS vs CCS, behaviorally

- Forward reductions of ρCCS configurations are CCS reductions
  $M \rightarrow M'$ implies $\alpha(M) \rightarrow \alpha(M')$
- Given a CCS reduction, this can be done by any ρCCS configuration mapped to it
  $P \rightarrow P'$ and $\alpha(M) = P$ implies $M \rightarrow M'$ and $\alpha(M') = P'$
  - History information has no impact on forward reductions

Valid configurations

- Not all the configurations are valid
- E.g., if the configuration contains a connector $k \prec k', k''$ then k' and k'' occur also as keys of a process, a memory or another connector
- Causality information should form a partial order
- A bit difficult to characterize syntactically valid configurations
- Semantic characterization: a configuration is valid iff it can be derived from a configuration of the form $\nu k\ k : P$

Parabolic Lemma

- Each computation is causally equivalent to a computation obtained by doing a backward computation followed by a forward computation
- Intuitively, I undo all what I have done and then compute only forward
  - Tries which are undone are not relevant
- Useful for proving the Causal consistency Theorem

Causal-consistent CCS in the literature

- In the literature there are two other causal-consistent reversible CCS
  - RCCS
    [Vincent Danos, Jean Krivine: Reversible Communicating Systems. CONCUR 2004]
    LTS based, histories attached to threads
  - CCSk
    [Iain C. C. Phillips, Irek Ulidowski: Reversing Algebraic Process Calculi. FoSSaCS 2006]
    LTS based, process is not consumed, part of it is just annotated as no more available

Are all those causal-consistent CCS equivalent?

- Yes!
- Reductions in ρCCS correspond to internal steps (τ moves) of the other approaches
- Essentially they provide different run time support definitions for the same language
- There exists a unique way to define a causal-consistent extension of a given language
  - Satisfying the expected properties
- Our approach is more easy to generalize

From ρCCS to Rhopi

- CCS is not expressive enough
- We want to consider more expressive languages
- We choose higher-order π-calculus
  - Allows processes to communicate

HOpi

- Syntax

$$P ::= a\langle P\rangle \mid a(X) \rhd P \mid P|Q \mid \nu a\, P \mid X \mid 0$$

- Higher-order communication
- Asynchronous calculus
- You can imagine structural congruence
- A reduction rule

$$a\langle P\rangle | a(X) \rhd Q \to Q\{^{P}/_{X}\}$$

Infinite behaviors

- HOπ can implement infinite behaviors
  - No need for operators for replication or recursion
- $Q = a(X) \rhd P|X|a\langle X\rangle$
  $Q \mid a\langle Q\rangle$ reduces to $P \mid \mathrm{Q} \mid a\langle Q\rangle$
- This allows one to generate an infinite amount of copies of P

How to make HOπ reversible?

- The main novelty is given by substitutions
- In ρCCS we can take the continuations from the configuration
- Here this is no more true
- From Q$\{^P/_X\}$ I cannot recover Q or P
- Not even Q if I know P
  - $P|X$, $X|P$, $P|P$, $X|X$ all produce the same result
- Not even P if I know Q
  - If Q does not contain X

Rhopi

- Syntax:

  $$M ::= k : P \mid [\mu, k\,] \mid k \prec k', k'' \mid M|M' \mid \nu u\ M \mid 0$$

  $$\mu ::= k : a\langle P\rangle | k' : a(X) \rhd Q$$

- Reduction rules:

  $$k : a\langle P\rangle | k' : a(X) \rhd Q \ \to \nu k''\ k'' : Q\{P/X\} | [\mu, k'']$$

  $$k'' : R | [\mu, k''] \ \rightsquigarrow \ \mu$$

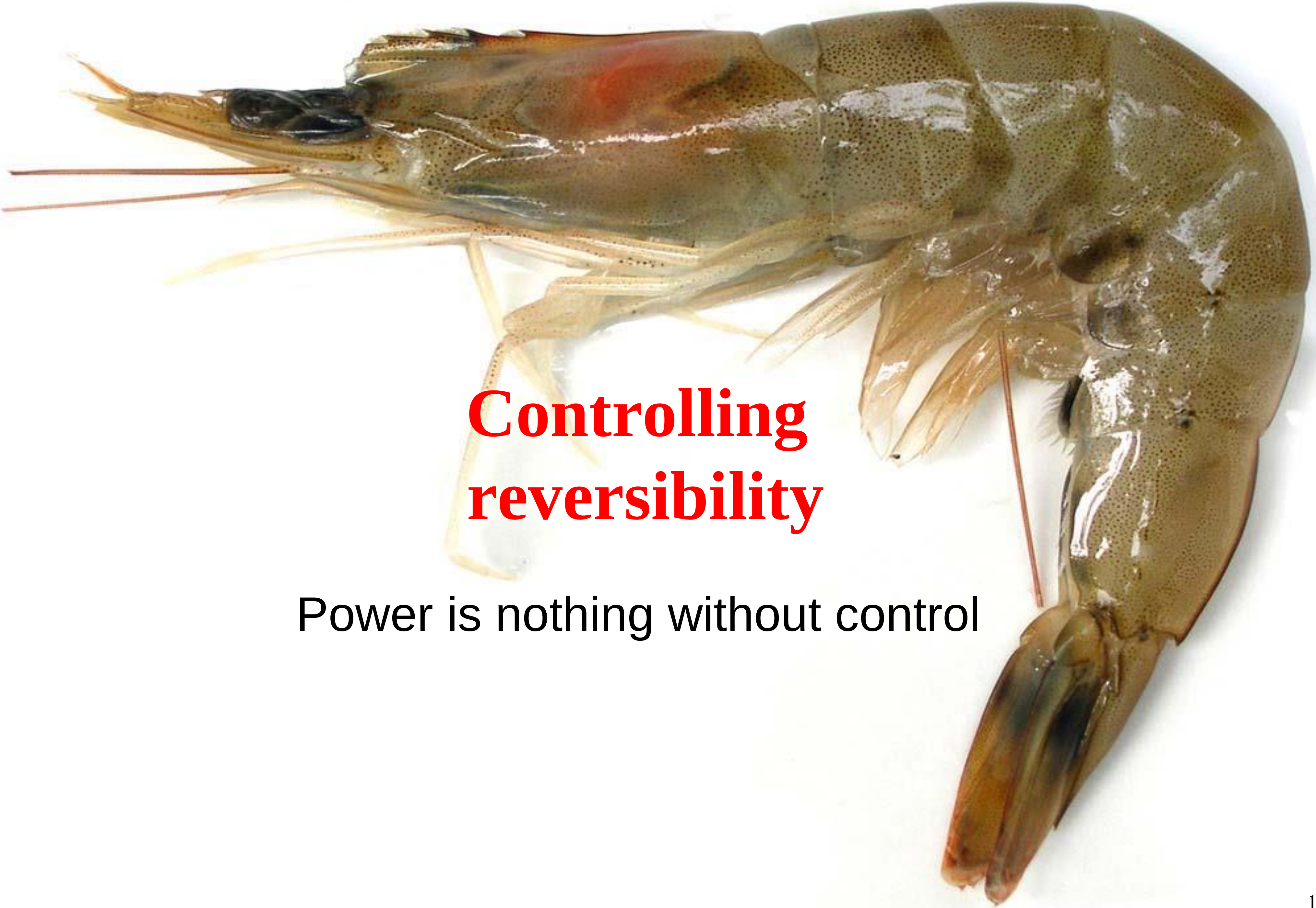
- A unique continuation since the calculus is asynchronous
- I store the whole configuration
  - Not really memory efficient
  - But it works, and provides a simple semantics
  - I may optimize it later

Restriction

- It seems we do not consider restriction
- Indeed, this is what we do
- We can do it!
- Try what happens with
  $k: a\left\langle \nu b\ c\langle b\langle Q\rangle .\, 0\rangle\right\rangle \mid k': a(X) \rhd X \mid k'': c(Y) \rhd Y$

Summarizing

- We have been able to define reversible CCS and HOπ
- Both causal consistent
- Using almost the same techniques
- But we are still at uncontrolled reversibility



Controlling reversibility

Power is nothing without control

Roll-π

- We want to use the **roll** operator to control reversibility in Rhopi
- We have to attach labels γ to some actions
  - We choose triggers
  - Since triggers have a continuation
- The challenge is to define the semantics of the **roll** operator
  - It involves an unbounded number of processes

Roll-π syntax

- $M ::= k : P \mid [\mu, k] \mid k \prec k', k'' \mid M|M' \mid \ \nu u\ M \mid 0$
- $P ::=$ $a\langle P\rangle \,\big|\, a(X) \rhd_{\gamma} P \,\big|\, P|Q \mid \nu a\ P \mid X \mid 0|\, roll\ \gamma \mid roll\ k$
- $\mu ::= k : a\langle P\rangle | k' : a(X) \rhd_{\gamma} Q$
- Now γ attached to triggers
- γ is a binder
- At run-time γ replaced by k

Roll-π semantics

- Little changes to the forward rule

$$k : a\langle P\rangle \mid k' : a(X) \rhd_{\gamma} Q \to$$
$$\nu k''\ k'' : Q\{P/X\}\{k''/\gamma\} \mid [\mu, k'']$$

- A new, complex, backward rule

$$\frac{k > M \quad complete(M \mid [\mu, k] \mid k' : roll\ k)}{M \mid [\mu, k] \mid k' : roll\ k \ \rightsquigarrow\ \mu \mid M \Lsh k}$$

- The two preconditions require to involve only processes which depend on k, and all of them
- We need to define the dependency relation

Exploiting causality

- Causal dependence: if in a term I have
  - $\left[k\colon a\langle P\rangle \middle| k'\colon a(X) \rhd_\gamma Q, k''\right]$ then $k > k''$ and $k' > k''$
  - $k \prec k', k''$ then $k > k'$ and $k > k''$
- $k > M$ if $k > h$ for all $h\colon P$, $[\mu, h]$ and $h \prec h', h''$ in M
- Completeness is essentially closure under consequences
- Completeness: if in a term I have
  - $\left[k\colon a\langle P\rangle \middle| k'\colon a(X) \rhd_\gamma Q, k''\right]$ then there is another occurrence of k''
  - $k \prec k', k''$ then there are other occurrences of k' and k''

Example

$$k: a\langle P\rangle \,\Big|\, k': a(X) \rhd_{\gamma} b(Y) \rhd roll\ \gamma \Big|\, k'': b\langle P'\rangle \rightarrow$$

$$\nu k'''[k: a\langle P\rangle \mid k': a(X) \rhd_{\gamma} b(Y) \rhd roll\ \gamma, k'''] \mid k''': b(Y)$$
$$\rhd roll\ k''' \mid k'': b\langle P'\rangle \ \rightarrow$$

$$\nu k'''\nu k''''[k: a\langle P\rangle \mid k': a(X) \rhd_{\gamma} b(Y) \rhd roll\ \gamma, k'''] \mid [k''': b(Y)$$
$$\rhd roll\ k''' \mid k'': b\langle P'\rangle, k''''] \mid k'''': roll\ k''' \ \rightsquigarrow$$

$$\nu k'''\nu k''''' k: a\langle P\rangle \,\Big|\, k': a(X) \rhd_{\gamma} b(Y) \rhd roll\ \gamma \Big|\, \color{red}{k'': b\langle P'\rangle}$$

Releasing resources

- Processes which are not dependent on k''' but are in memories dependent on k''' can be seen as resources taken by the computation from the environment
- They have to be restored in case of **roll** k'''
- This is done by $M \Lsh k'''$