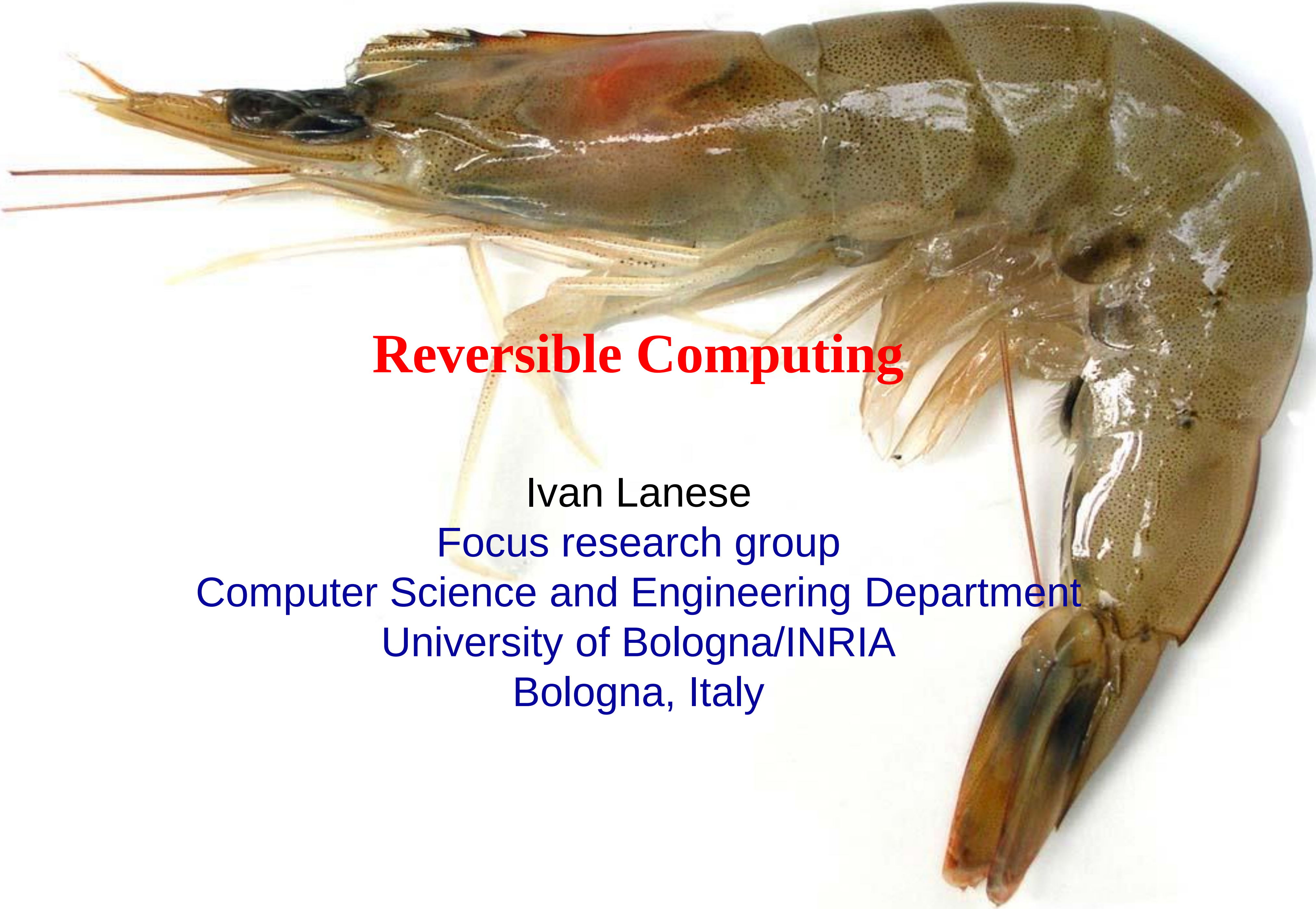


Reversible Computing

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

Roll-π reminder

- Controlled version of rhopi
- Based on operator **roll** γ
- Semantics defined by the rule below

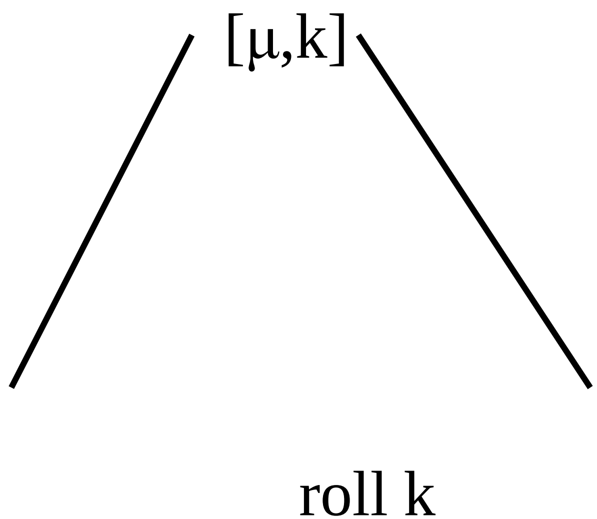
$$\frac{k > M \quad complete(M \,|\, [\mu, k] \,|\, k' : roll\ k)}{M \,|\, [\mu, k] \,|\, k' : roll\ k \ \rightsquigarrow \mu \,|\, M \Lsh k}$$

Is roll-π a controlled rhopi?

- Let φ be a function that removes all γ and replaces all **roll**s with 0
  - Maps roll-π configurations to rhopi configurations
- $M \rightarrow M'$ (controlled) iff $\varphi(M) \rightarrow \varphi(M')$ (uncontrolled)
- If $M \rightsquigarrow M'$ (controlled) then $\varphi(M) \rightsquigarrow^{+} \varphi(M')$ (uncontrolled)
  - The opposite implication holds only if a suitable **roll** exists

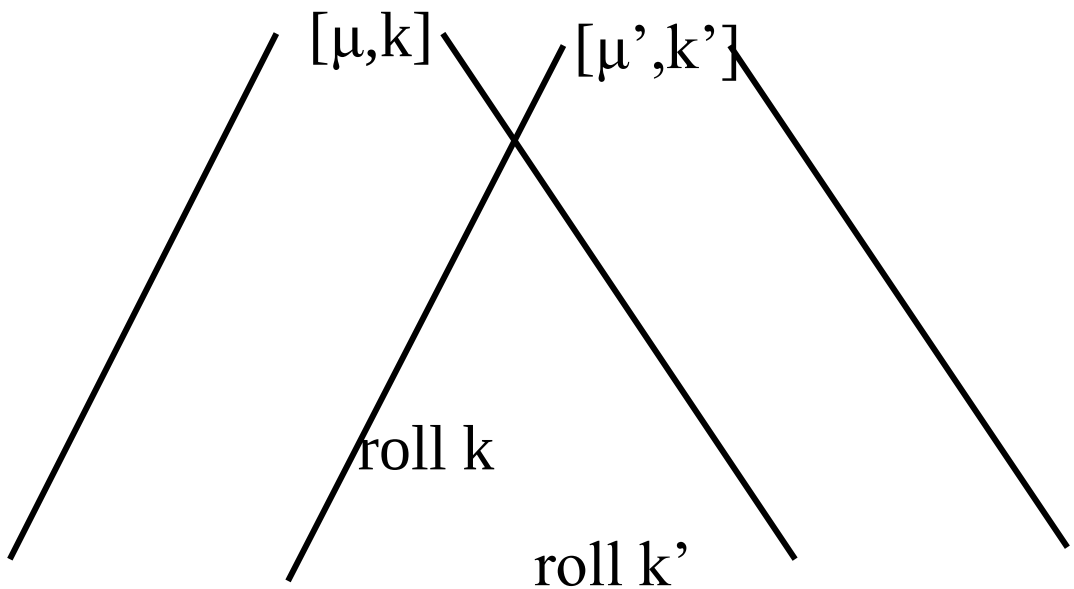
A graphical interpretation of **Roll**

- One can see the processes involved in a rollback as the tree of consequences of the key of the roll



Roll and concurrency

- Two **roll**s may interfere



- Executing one **roll** removes the other
- In a concurrent setting I would be able to execute both of them

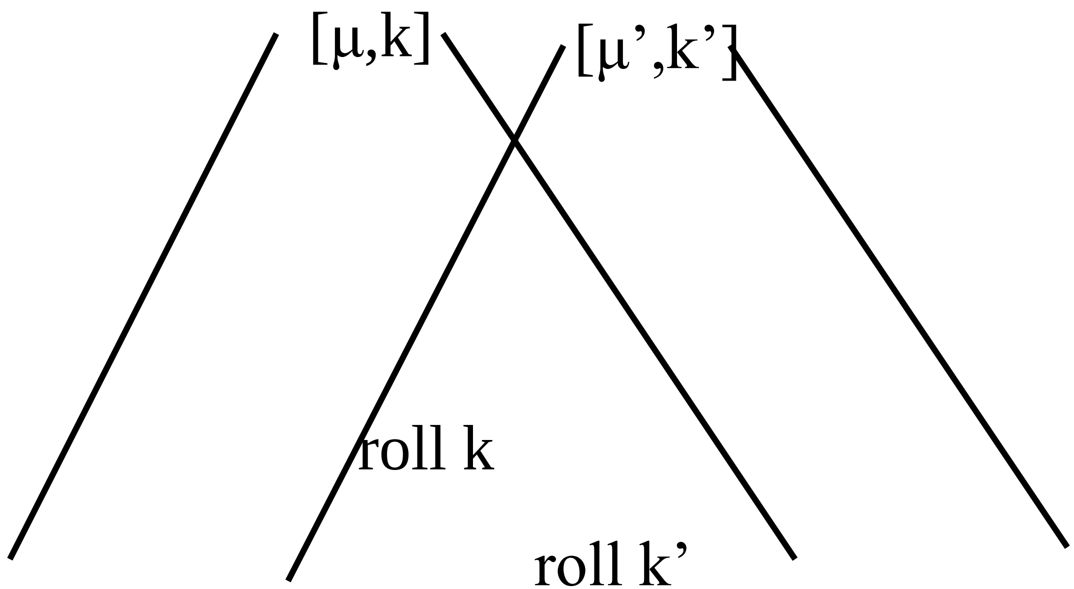
Concurrent semantics for **Roll**

- I can get the power of concurrent **roll**s with a simple trick
- Two steps rollback
  - First, I mark the target memory
  - Second, I execute the **roll**

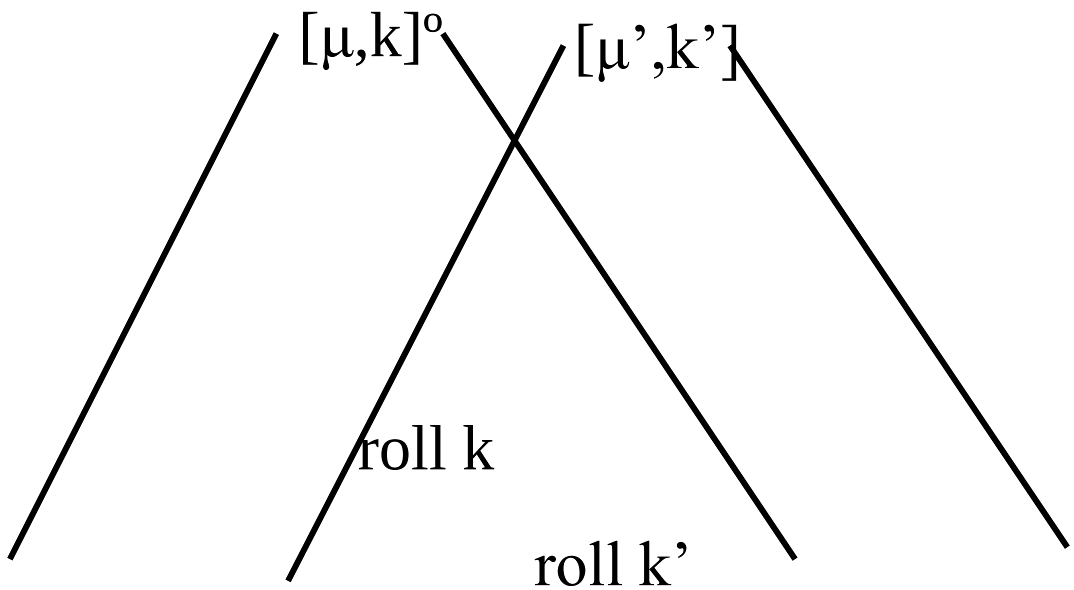
$$[\mu, k] | k' : roll\ k \rightsquigarrow [\mu, k]^\circ | \ k' : roll\ k$$

$$\frac{k > M \quad complete(M | [\mu, k]^\circ)}{M | [\mu, k]^\circ \rightsquigarrow \mu | M \lightning k}$$

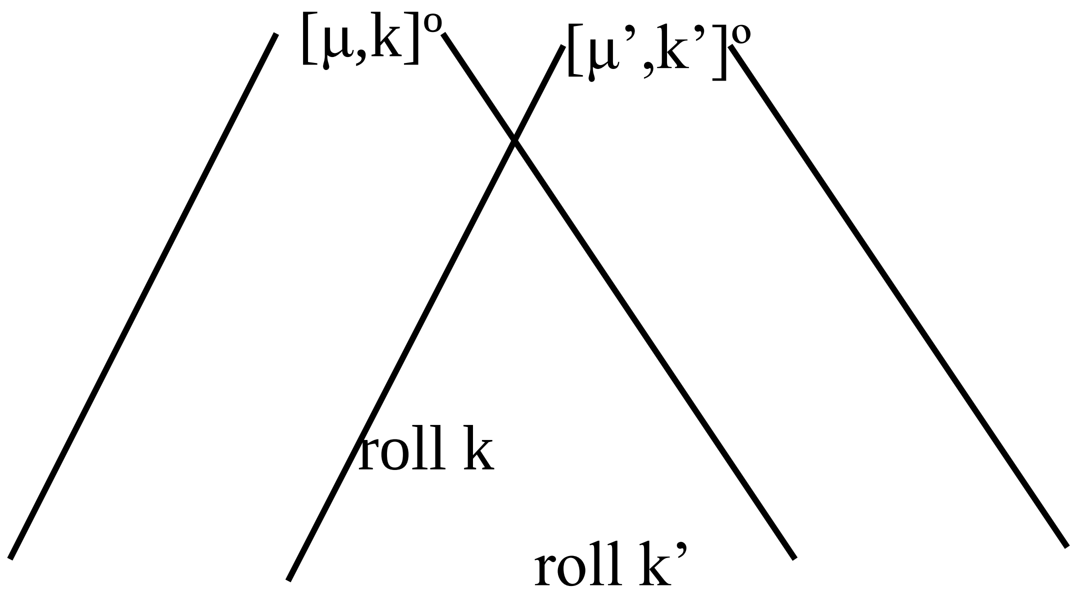
Executing two concurrent **roll**s



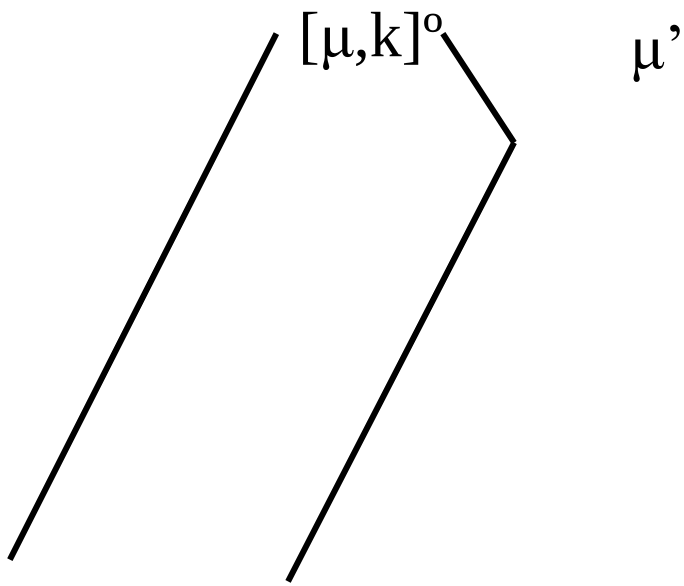
Executing two concurrent **roll**s



Executing two concurrent **roll**s



Executing two concurrent **roll**s



Executing two concurrent **roll**s

μ μ'

Going towards an implementation

- The rule defining the behavior of **roll** is not easy to implement
  - It involves an unbounded number of processes
- This semantics is a specification, not a guide to the implementation
- We can define a lower level semantics nearer to an implementation
- The low level semantics and the concurrent semantics are equivalent

A lower level semantics

- Essentially a distributed algorithm based on message passing
- The marked memory sends messages “freeze” to all the descendants
  - The descendants forward the messages
  - If the descendant is a memory, the process(es) depending on the **roll** key are frozen
- When the message reaches a leaf, the leaf suicides by notifying its ancestors
  - If the leaf is a memory, non frozen processes are released
- The algorithm terminates when the marked memory is reached

Lower level semantics features

- Only binary interactions
- Easy to implement
- Indeed, we implemented it in Maude
- **Roll** execution is no more atomic
  - Loss of atomicity causes no fake interactions
  - But a **roll** execution may not terminate
- Difficult to find a correspondence with the sequential semantics
  - Would require global locks

Specifying alternatives

No divergence please

Specifying alternatives in croll-π

- In roll-π every process featuring an executable **roll** has a divergent computation
- We want to give to the programmer tools to avoid this
- We use alternatives
- We add the simplest possible form of alternative
  - If something is simple and works, it is probably good

Messages with alternative

- We attach alternatives only to messages
- Instead of messages $a\langle P\rangle$ we use messages with alternative
  - $a\langle P\rangle\%0$: try $a\langle P\rangle$, then stop trying
  - $a\langle P\rangle\%b\langle Q\rangle\%0$: try $a\langle P\rangle$, then $b\langle Q\rangle$, then stop trying
- If the message with alternative is the target of the **roll**, it is replaced by its alternative
- Very little change to the syntax
- Also the semantics is very similar
- The expressive power increases considerably

Croll-π syntax

- $M ::= k : P \mid [\mu, k] \mid k \prec k', k'' \mid M|M' \mid \nu u\, M \mid 0$
- $P ::= a\langle P\rangle\%A \mid a(X) \rhd_\gamma P \mid P|Q \mid \nu a\, P \mid X \mid 0$
  $\mid roll\ \gamma \mid roll\ k$
- $\mu ::= k : a\langle P\rangle\%A \mid k' : a(X) \rhd_\gamma Q$
- $A ::= 0 \mid a\langle P\rangle\%0$
- Now messages have alternatives

Croll-π semantics

- Little changes to the forward rule

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

- Little changes to the backward rule

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

- Function xtr replaces messages with alternative with their alternative
- $xtr(a\langle P\rangle\%A) = A$

Arbitrary alternatives

- We only allow 0 and messages with 0 alternative as alternatives?
  - Is this enough?
- We can encode arbitrary alternatives
- $[\![a\langle P\rangle\%Q]\!] = \nu c\ a\langle P\rangle\%c\langle Q\rangle\%0 \mid c(X) \rhd X$
- Q can even have alternatives
- $a_1\langle P_1\rangle\%a_2\langle P_2\rangle\% \ldots \%a_n\langle P_n\rangle\%0$
  - I try different options
  - By choosing $a_1, \ldots, a_n = a$ and $P_1, \ldots, P_n = P$ I try the same possibility n times before giving up

Endless retry

- I can retry the same alternative infinitely many times
  - As in roll-π
- $[\![a\langle P\rangle]\!] = \nu c\ Q \mid a\langle [\![P]\!]\rangle \% c\langle Q\rangle$
- $Q = c(Z) \rhd Z \mid a\langle [\![P]\!]\rangle \% c\langle Z\rangle$
- As for replication, we can encode infinite behaviors using process duplication

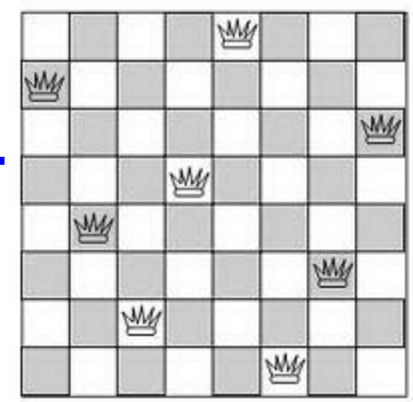
Triggers with alternative

- We can attach alternatives to triggers instead of messages
- $[\![\big(a(X) \rhd_\gamma Q\big)\% b\langle Q'\rangle\%0]\!] =$
  $\nu c\ \nu d\ c\langle 0\rangle\% d\langle 0\rangle\%0 \mid \big(c(Y) \rhd_\gamma a(X) \rhd [\![Q]\!]\big) \mid$
  $(d(Z) \rhd b\langle [\![Q']\!]\rangle\%0)$
- Triggers with alternative make the framework more symmetric
- I cannot mix triggers with alternative and messages with alternative

Expressive power

- Do alternatives increase the expressive power?
- Yes!
- We can prove this using encodings
- We can encode roll-π into croll-π
  - Using endless retry
- We cannot do the opposite, preserving
  - Existence of a backward reduction
  - Termination

The 8 queens



$$
\begin{aligned}
Q_i \;\triangleq\; & (act_i(Z) \triangleright p_i\langle i, 1\rangle \div \ldots \div p_i\langle i, 8\rangle \div f_i\langle \mathbf{0}\rangle \div \mathbf{0} \mid \\
& \quad (p_i(\mathbf{x_i}) \triangleright_{\gamma_i} !c_i\langle \mathbf{x_i}\rangle \div \mathbf{0} \mid act_{i+1}\langle \mathbf{0}\rangle \mid f_{i+1}(Y) \triangleright \mathsf{roll}\ \gamma_i \mid \\
& \qquad \textstyle\prod_{j=1}^{i-1} c_j(\mathbf{y_j}) \triangleright \mathtt{if}\ err(\mathbf{x_i}, \mathbf{y_j})\ \mathtt{then}\ \mathsf{roll}\ \gamma_i))
\end{aligned}
$$

$$
err((x_1, x_2), (y_1, y_2)) \;\triangleq\; (x_1 = y_1 \vee x_2 = y_2 \vee |x_1 - y_1| = |x_2 - y_2|)
$$

- ! denotes replication
  - We know we can encode it
- Compact and concurrent implementation
- A more concurrent but less efficient implementation also exists