Simulating Biological Systems in the Stochastic Pi-Calculus

Andrew Phillips Luca Cardelli

28th July 2004

Microsoft Research
7 JJ Thomson Avenue
Cambridge, UK

Introduction

- Ongoing Experiment:
  - Use process calculi to model biological systems
- Features of process calculi:
  - *Compositional* modelling, analysis and simulation of systems.
- Potential Benefits:
  - *Understand* complex systems by decomposing them into simpler subsystems.
  - *Analyse* properties of subsystems using established theory.
  - *Predict* behaviour of subsystems by running stochastic simulations.
  - Predict properties and behaviour of *composed* systems.
- Pi-calculus: one of the simplest and most well-studied calculi.

Outline

- Graphical Pi-Calculus
- Chemical Reactions
- Gene Regulation
- Simulator

Pi-Calculus

- Syntax:

$P, Q ::=$	$\nu n\, P$	Restriction		$\Sigma ::=$	$\mathbf{0}$	Null
$\mid$	$P \mid Q$	Parallel		$\mid$	$\pi.P + \Sigma$	Action
$\mid$	Σ	Summation		$\pi ::=$	$x\langle n\rangle$	Output
$\mid$	$!\pi.P$	Replication		$\mid$	$x(m)$	Input

- Semantics:

$$Q \equiv P \wedge P \longrightarrow P' \wedge P' \equiv Q' \quad \Rightarrow \quad Q \longrightarrow Q'$$

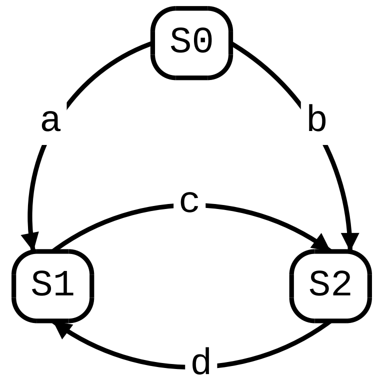
$$P \longrightarrow P' \quad \Rightarrow \quad \nu n\, P \longrightarrow \nu n\, P'$$

$$P \longrightarrow P' \quad \Rightarrow \quad P \mid Q \longrightarrow P' \mid Q$$

$$(x\langle n\rangle.P + \Sigma) \mid (x(m).Q + \Sigma') \longrightarrow P \mid Q_{\{n/m\}}$$

Graphical Pi-Calculus

- We want an intuitive representation for pi-calculus. Like FSMs...

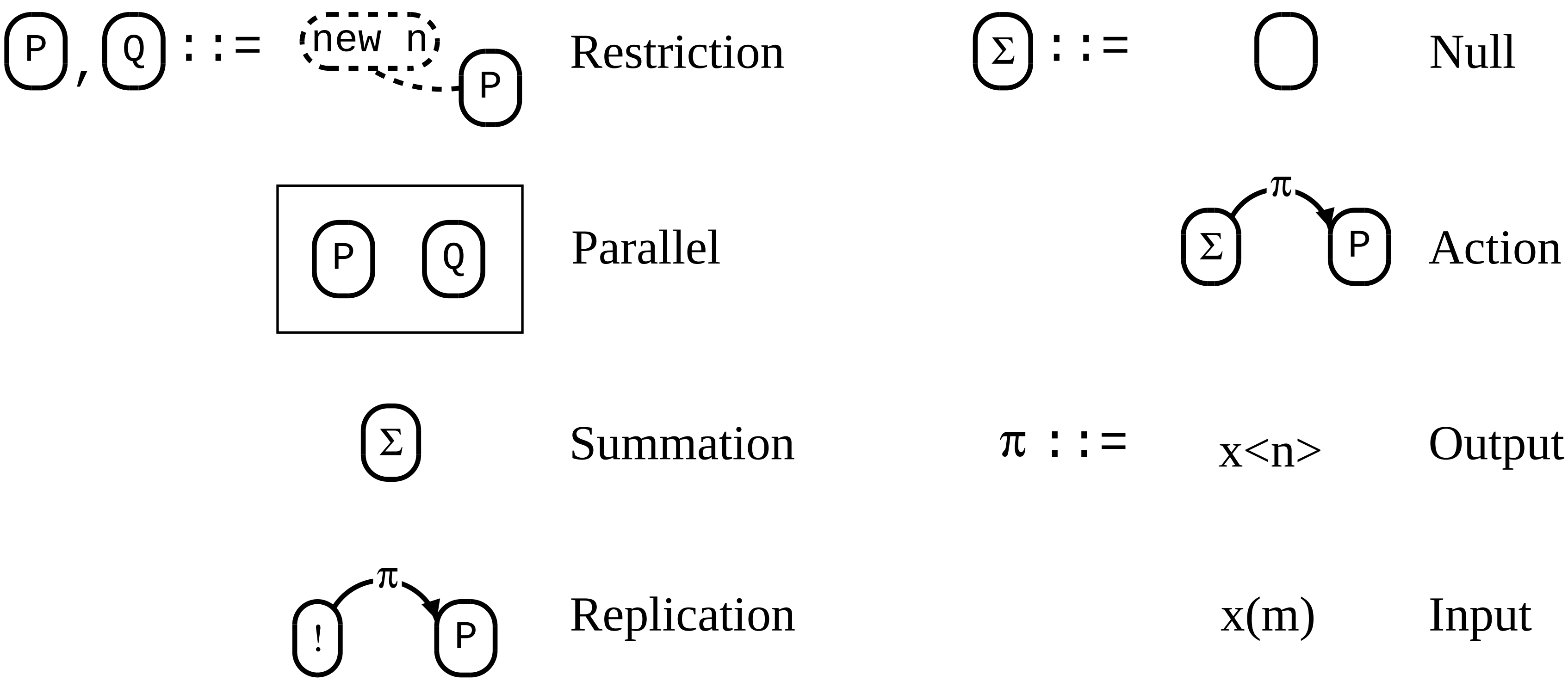


- But with all the features of pi: compositionality, restriction, communication, replication.

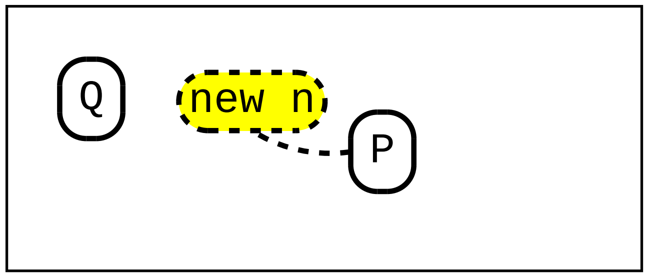
- Should be a 1-1 correspondence between graphics and text

- NO NEW THEORY

Graphical Syntax

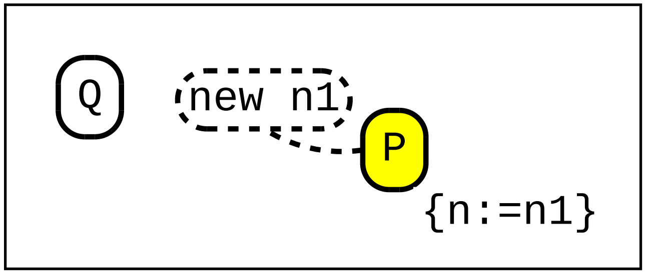


Graphical Semantics: Restriction



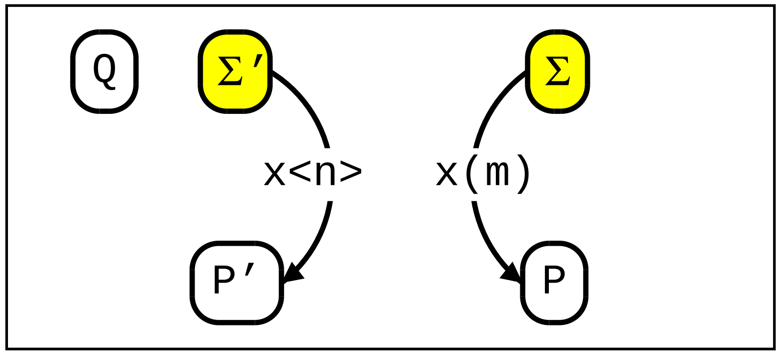
➢ Restriction creates a fresh name inside a given process.

Graphical Semantics: Restriction



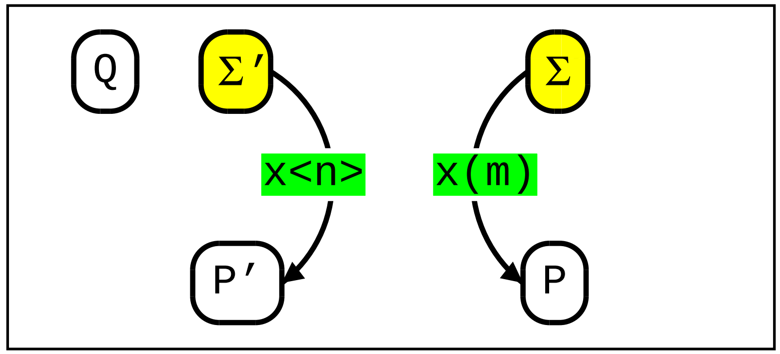
➢ The name n is replaced with a fresh name $n1$ that is unknown to Q.

Graphical Semantics: Communication



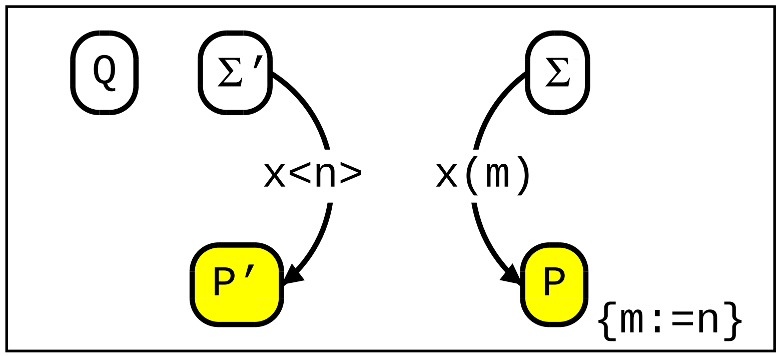
➢ Two parallel summations can interact on a common channel.

Graphical Semantics: Communication



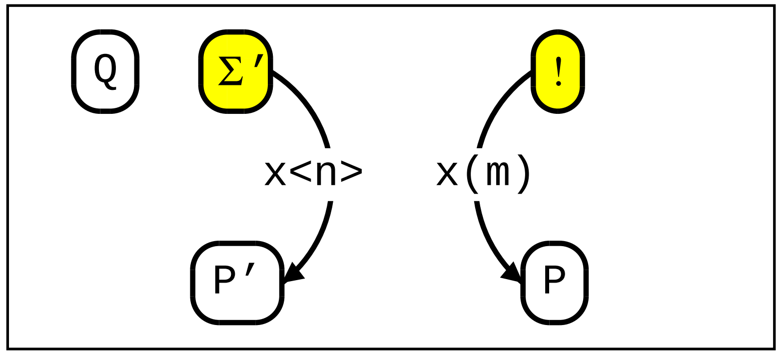
➢ An output $x\langle n\rangle$ can send a message n on channel x to an input $x(m)$.

Graphical Semantics: Communication



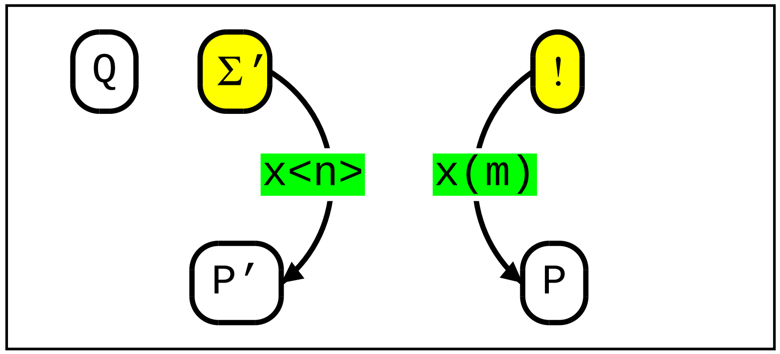
➢ Message n is assigned to m in process P'.

Graphical Semantics: Replication



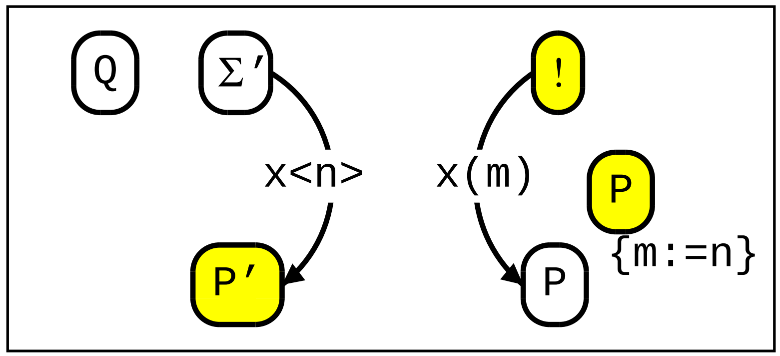
➢ A replicated input can spawn a clone of a process.

Graphical Semantics: Replication



➢ An output $x\langle n\rangle$ can send a message n to a replicated input $!x(m)$.

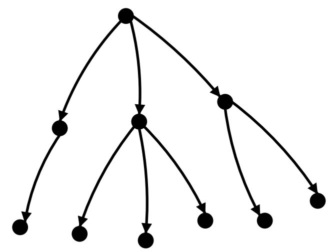
Graphical Semantics: Replication



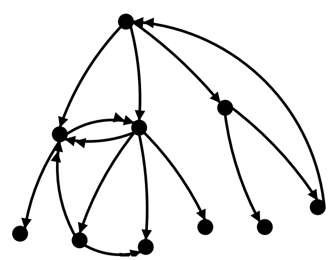
➢ A clone of P is spawned and message n is assigned to m in the clone.

Trees vs Graphs

- So far, the syntax of a graphical pi process is a *tree* of nodes.

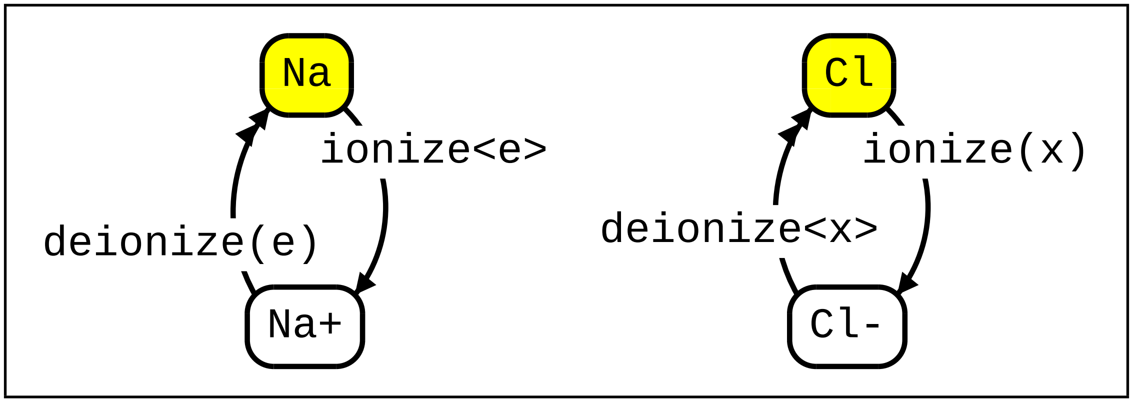


- But we really want a graph, like FSMs... Fortunately, *links* between nodes in a tree can be *encoded*.



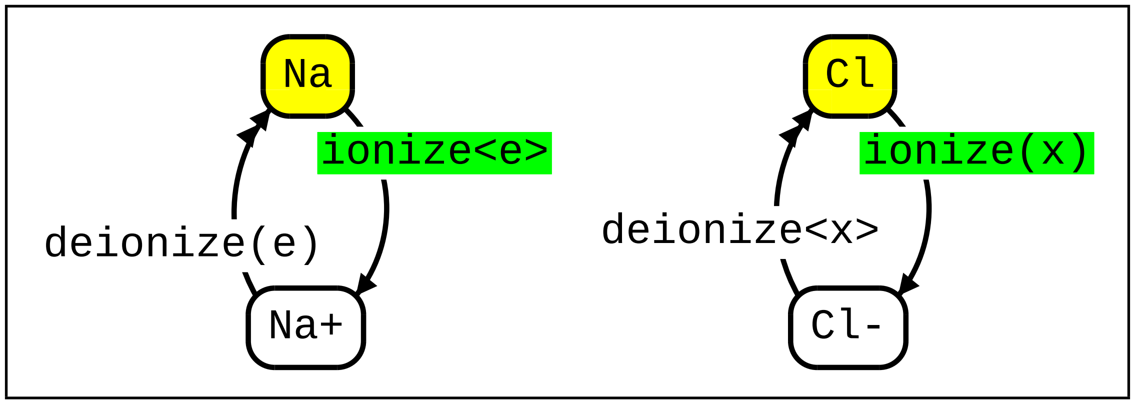
- The result is an arbitrary graph with two kinds of edges.

Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$



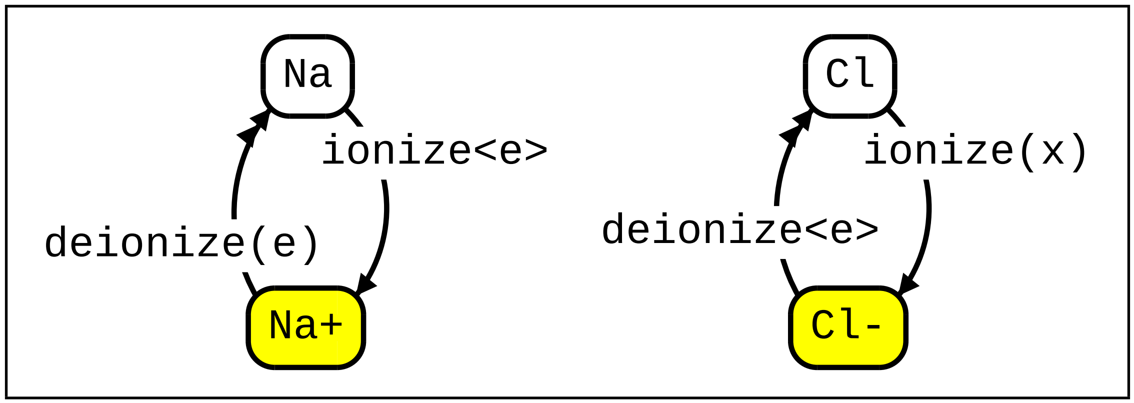
- Na can ionize Cl by sending its electron, with rate $100s^{-1}$
- Cl^- can deionize Na^+ by sending its electron, with rate $10s^{-1}$
- State names are merely *annotations*

Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



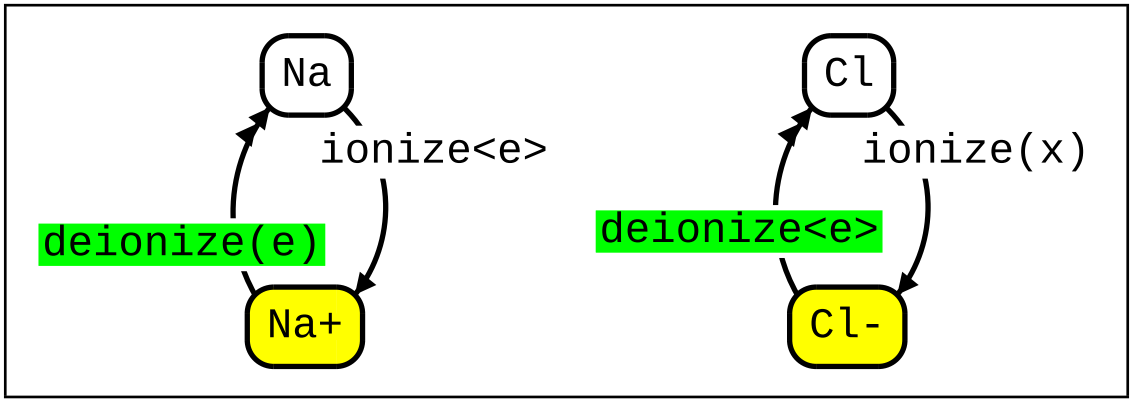
- Na can ionize Cl by sending its electron on the $ionize$ channel

Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



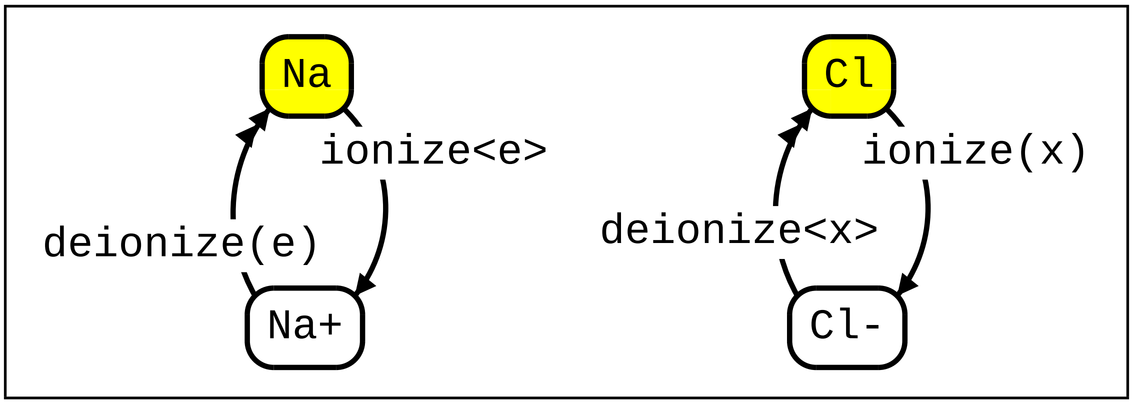
- Na^{+} is positively charged and Cl^{-} is negatively charged

Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$



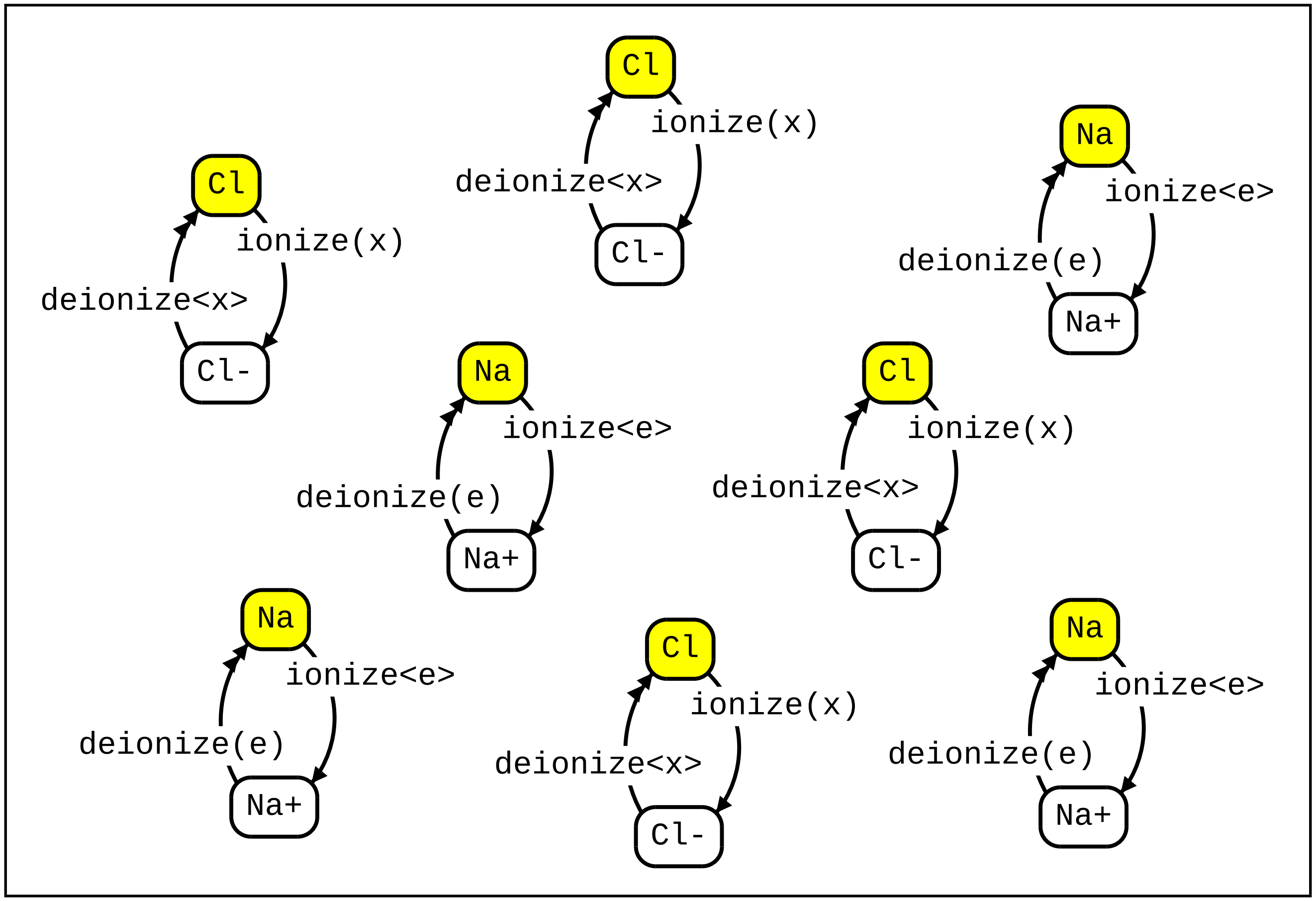
- Cl^- can deionize Na^+ by sending its electron on the *deionize* channel

Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$

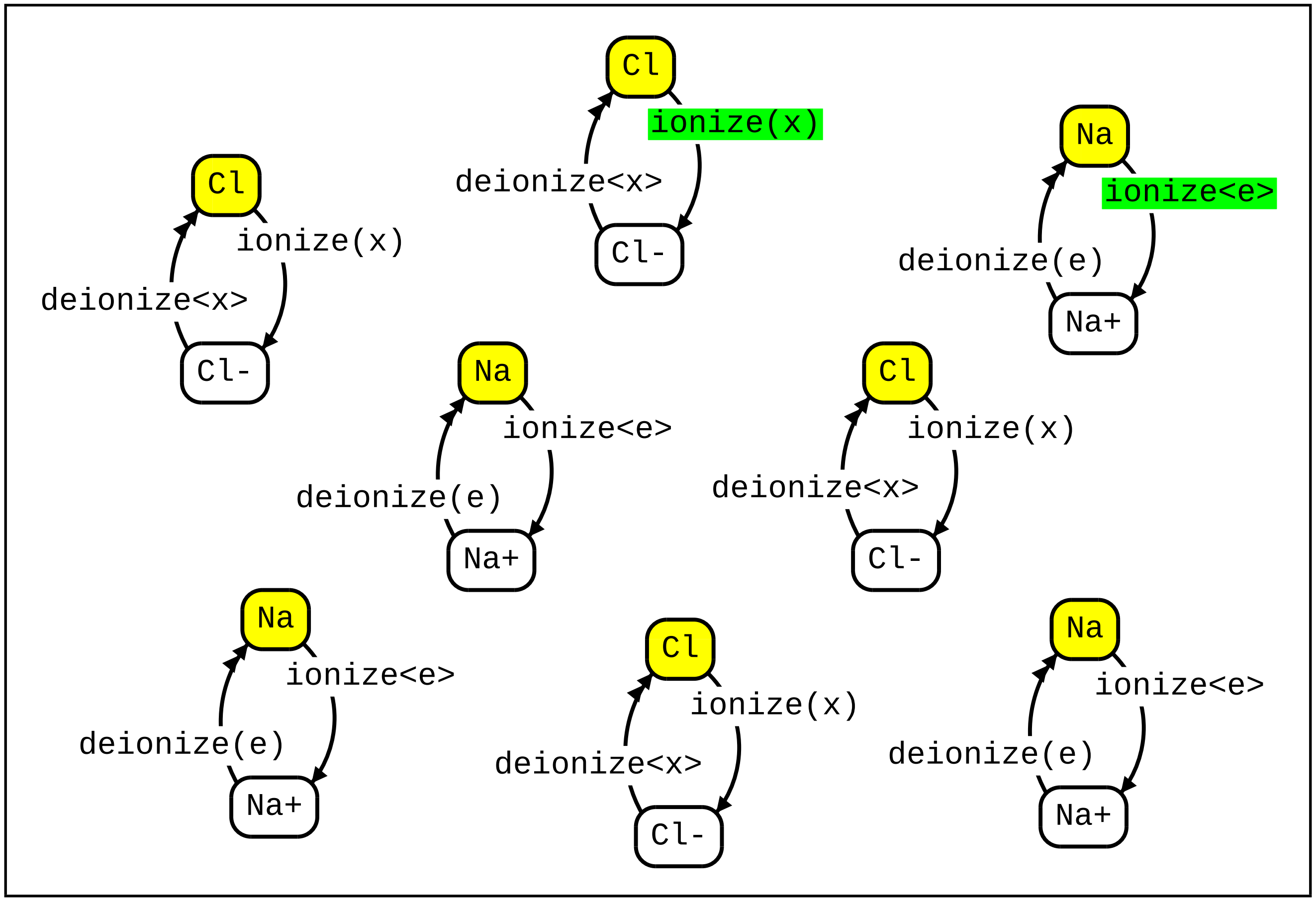


➢ Na and Cl are no longer charged

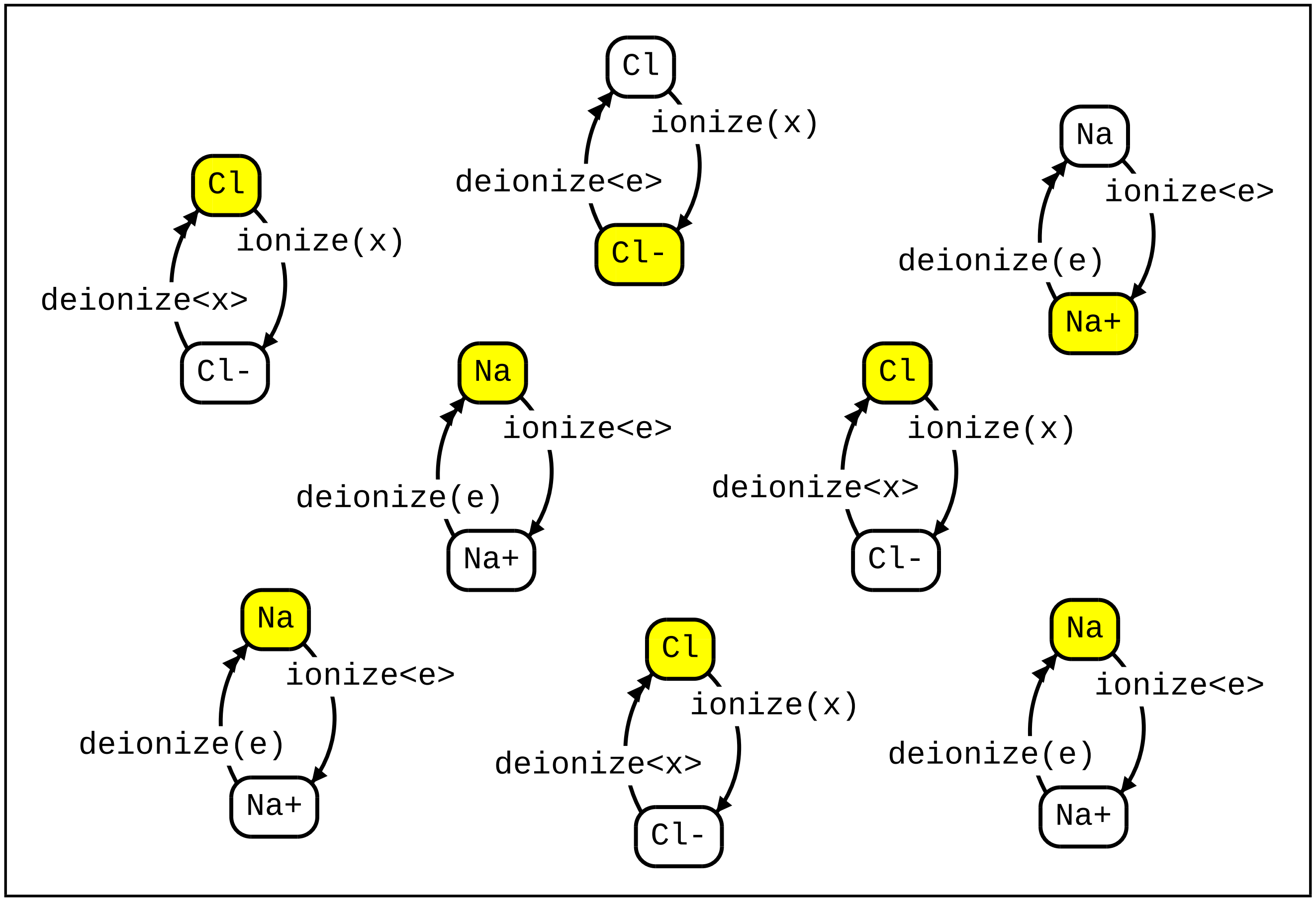
Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



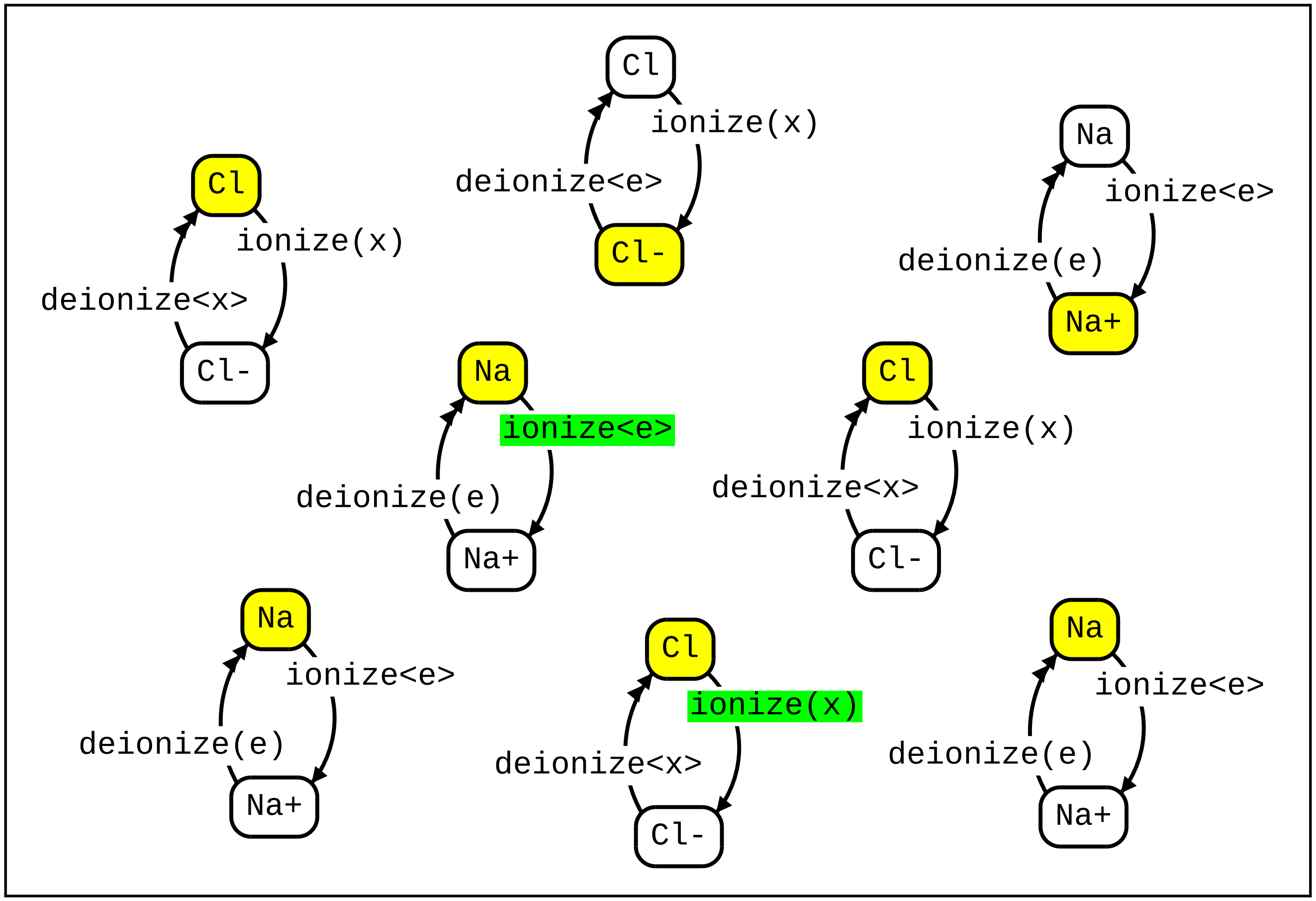
Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



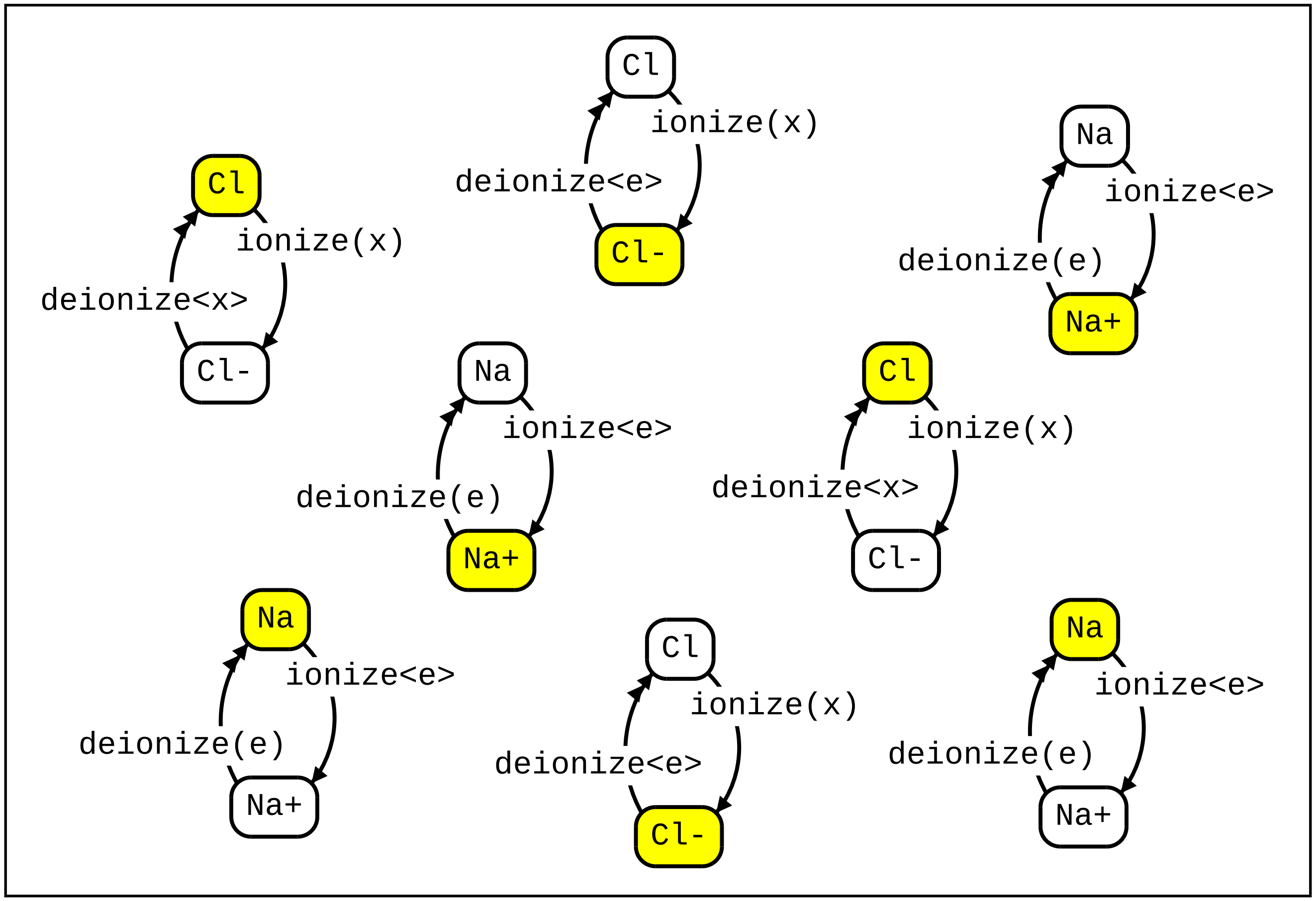
Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



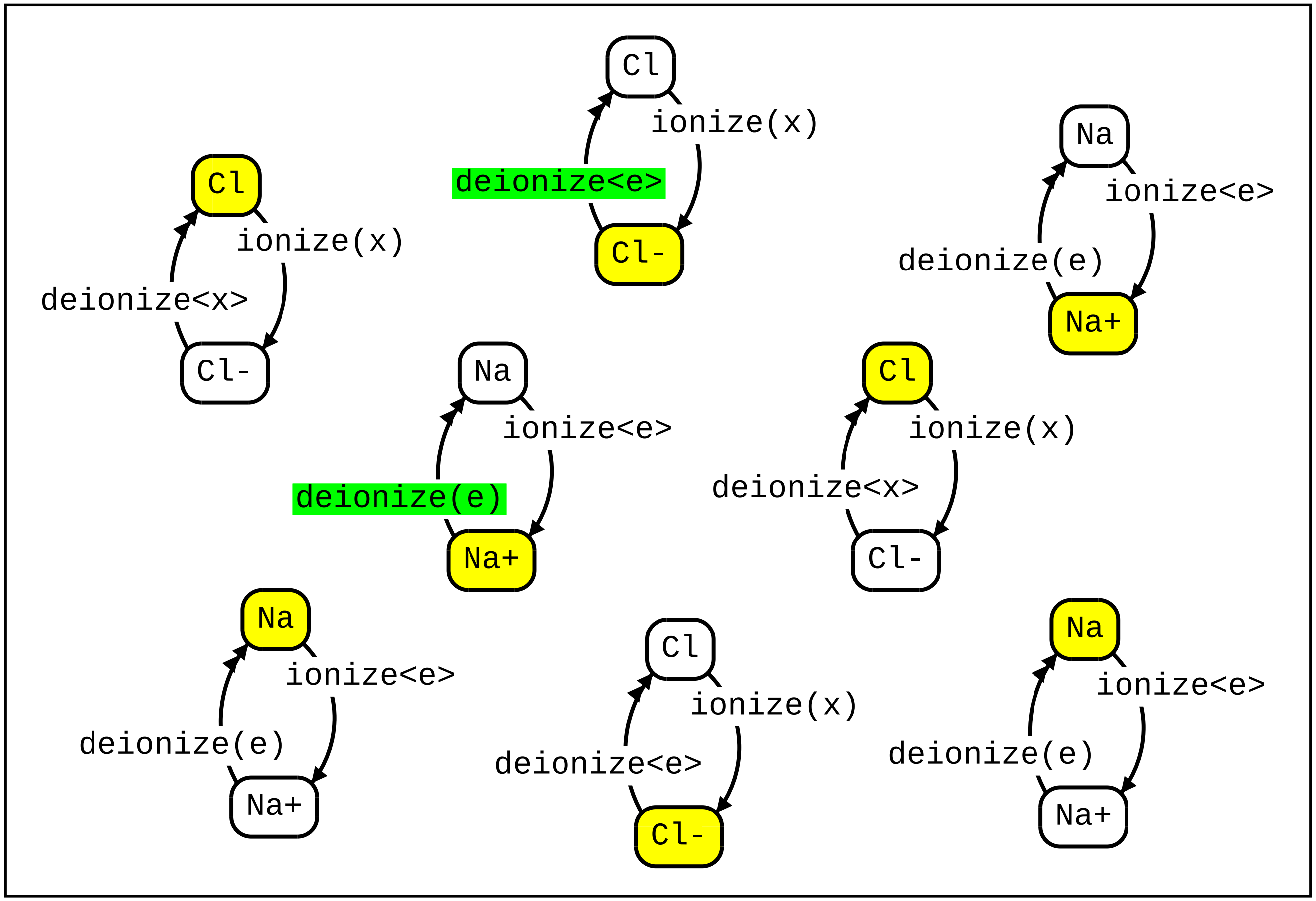
Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



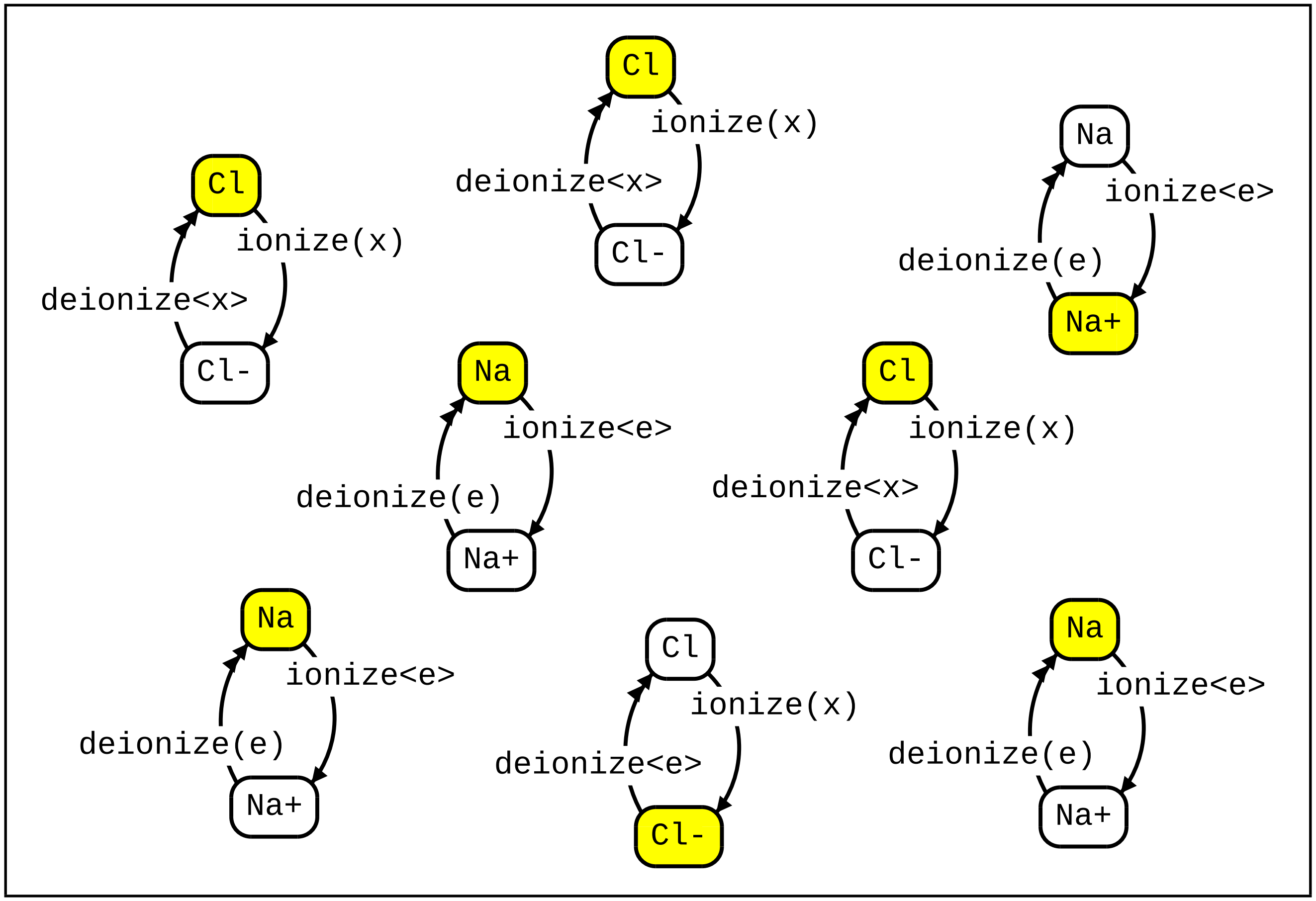
Ionization: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$



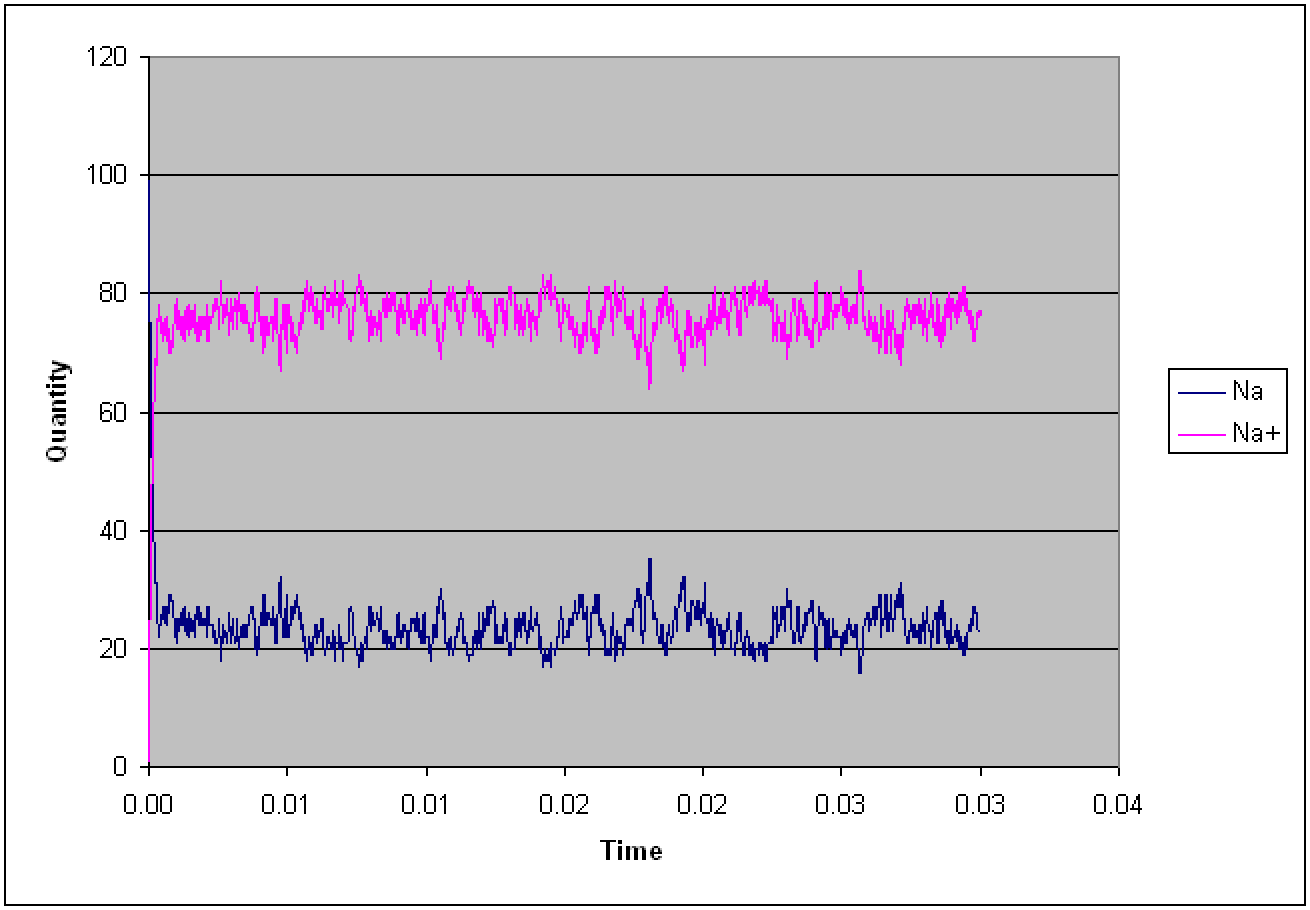
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$



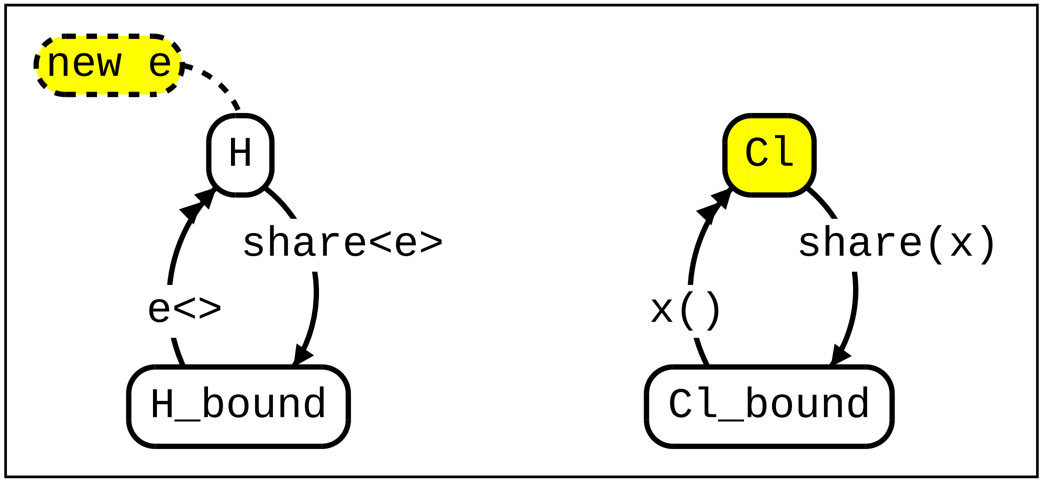
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$



Virtual Experiment: $Na + Cl \rightleftharpoons Na^{+} + Cl^{-}$

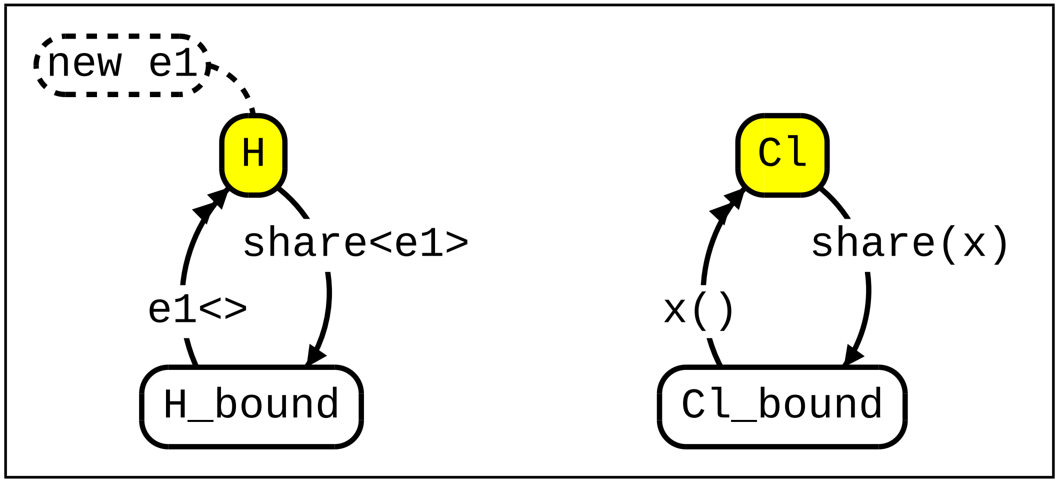


Covalent Bonding: $H + Cl \rightleftharpoons HCl$



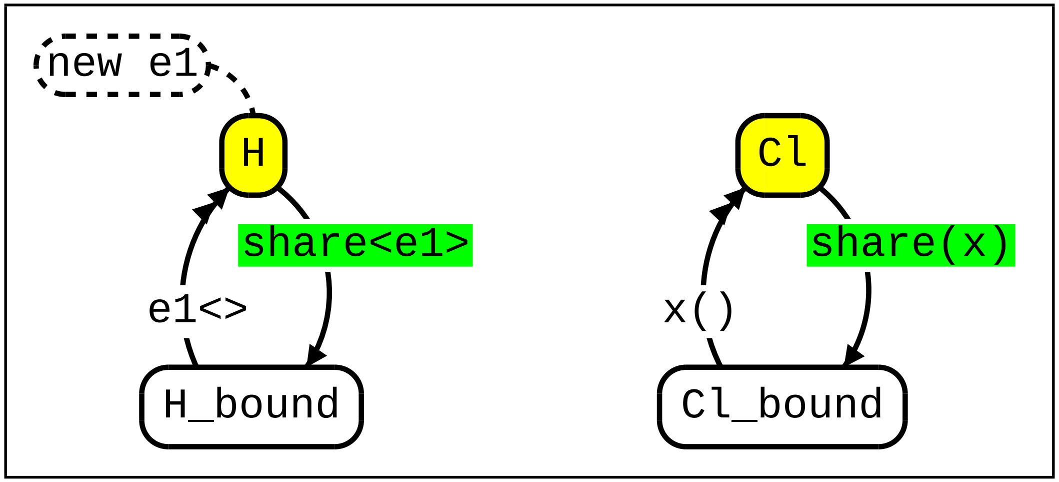
- H has a *private* electron.
- H can share its electron with Cl to form a covalent bond with rate $100s^{-1}$
- HCl can break its private bond with rate $10s^{-1}$

Covalent Bonding: $H + Cl \rightleftharpoons HCl$



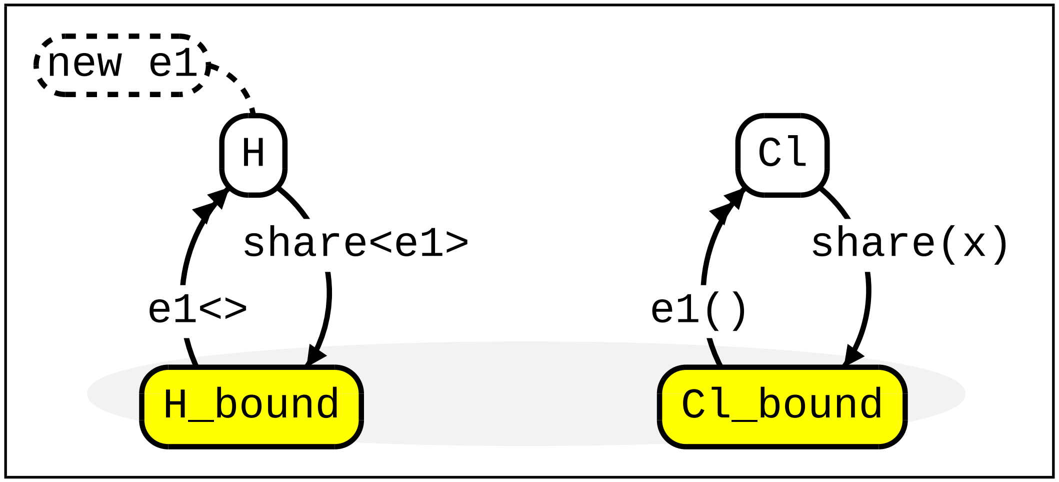
➢ H has a private electron $e1$ that is not accessible from outside.

Covalent Bonding: $H + Cl \rightleftharpoons HCl$



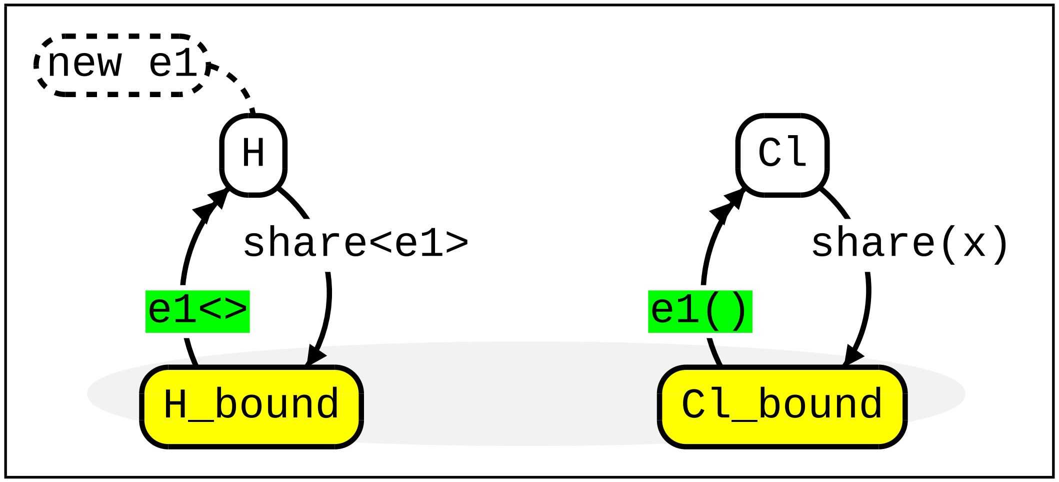
➢ H can share its electron with Cl on the $share$ channel.

Covalent Bonding: $H + Cl \rightleftharpoons HCl$



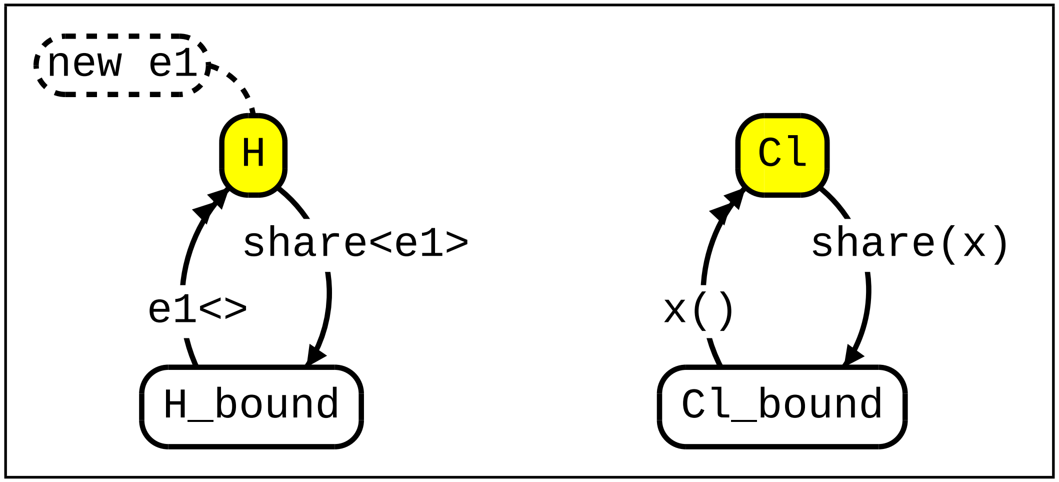
➢ H and Cl share a private electron, to form HCl.

Covalent Bonding: $H + Cl \rightleftharpoons HCl$



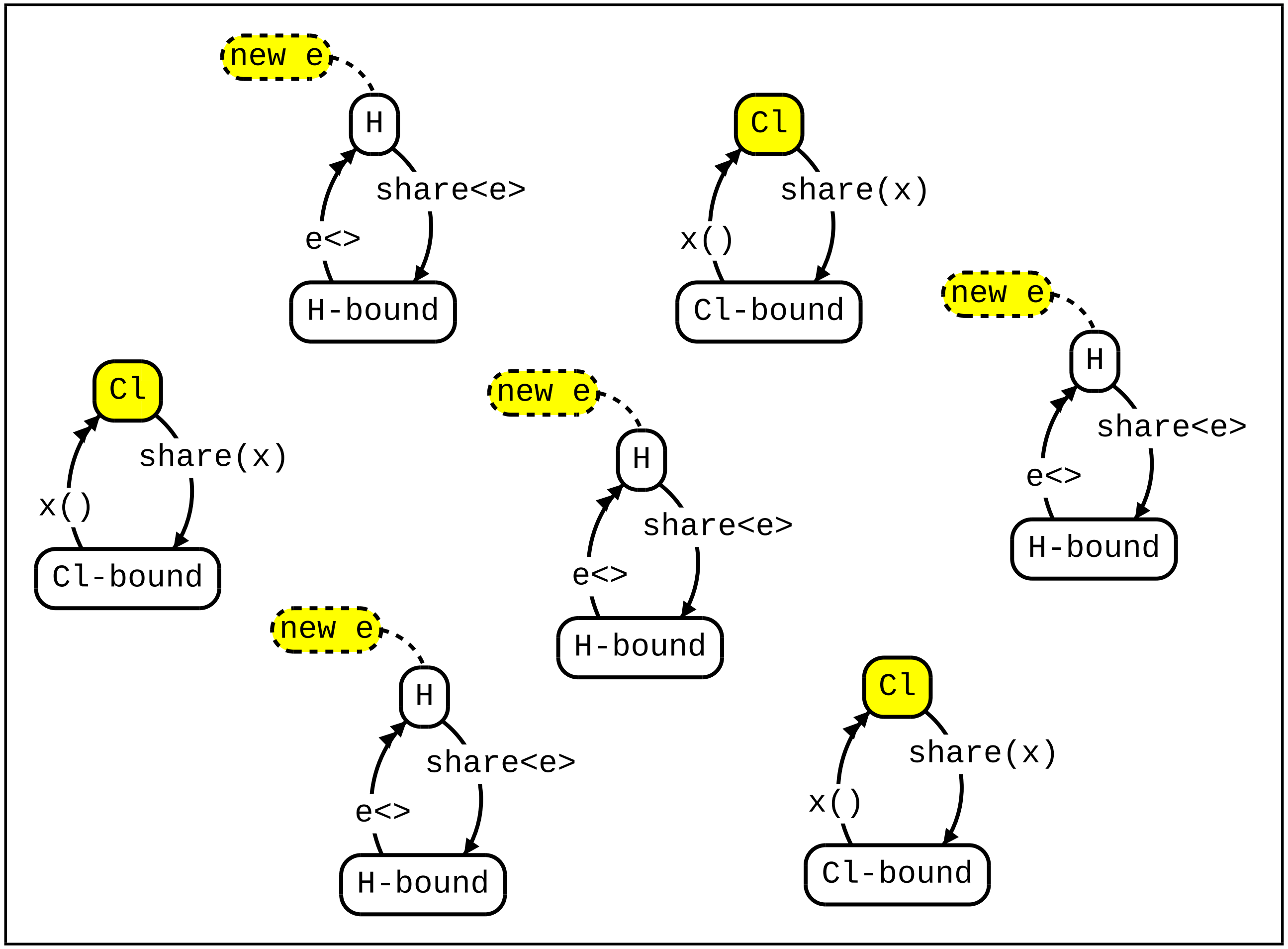
- HCl can break its private bond with rate $10s^{-1}$

Covalent Bonding: $H + Cl \rightleftharpoons HCl$

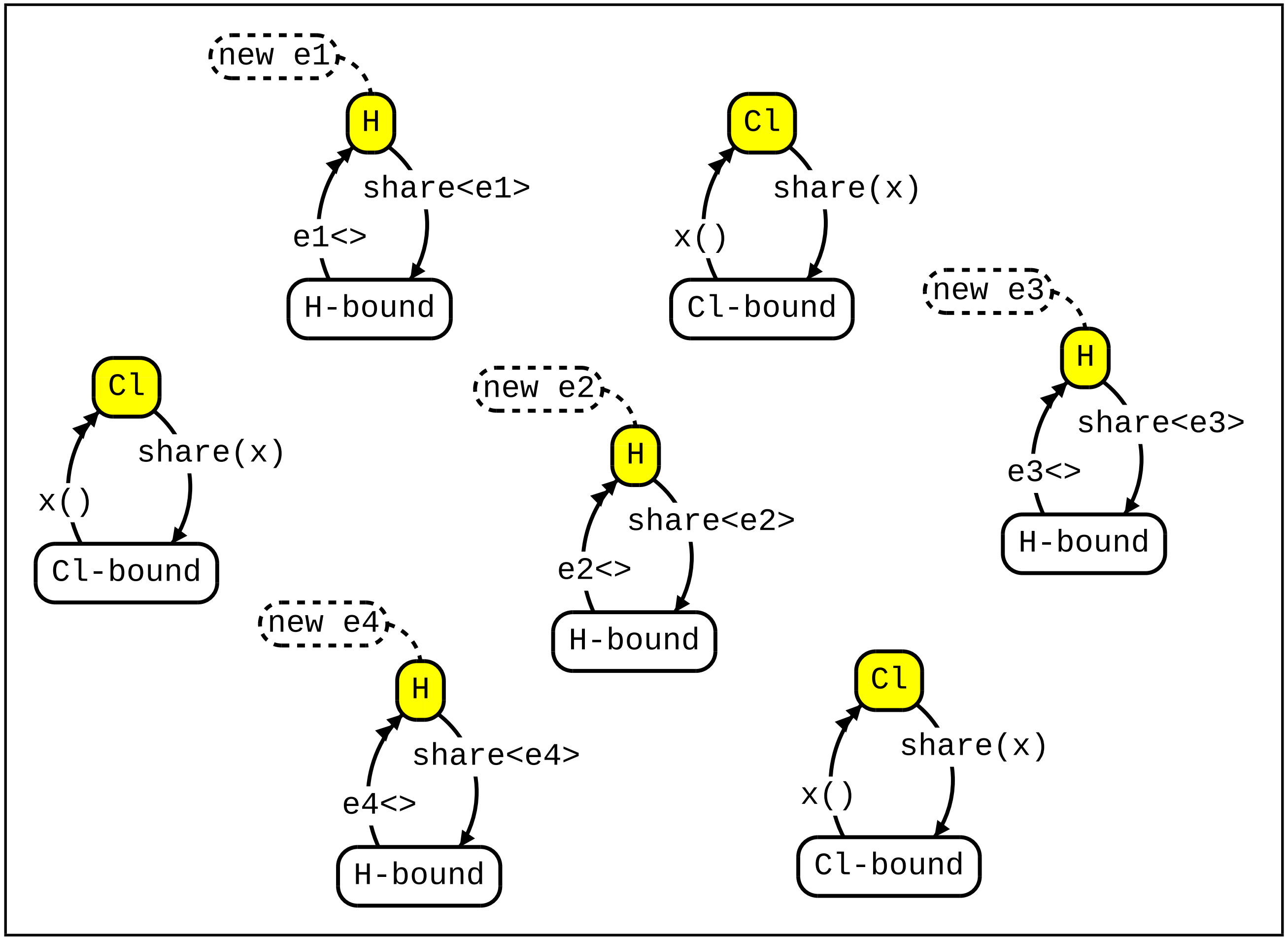


➢ H and Cl are no longer bound

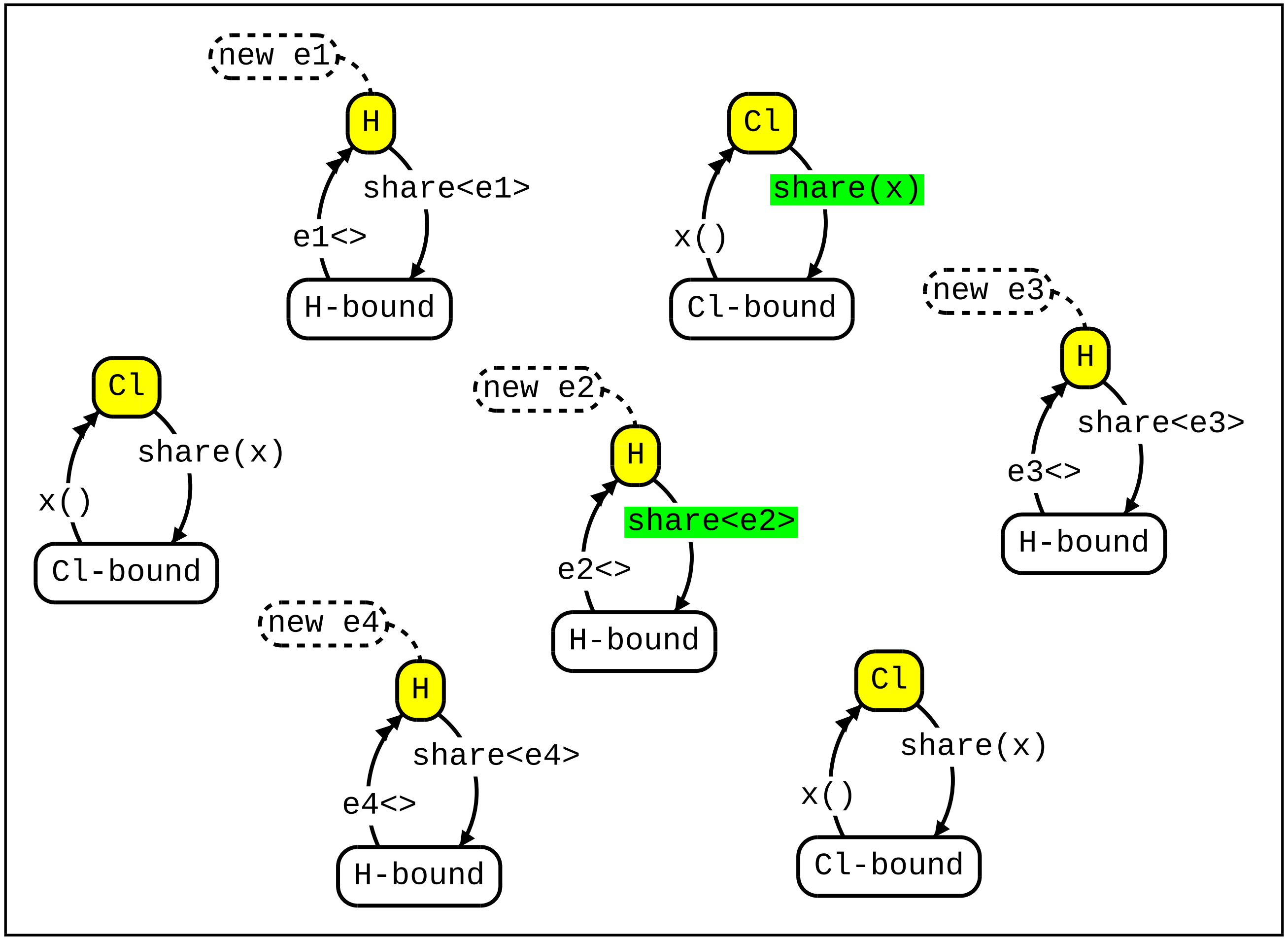
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



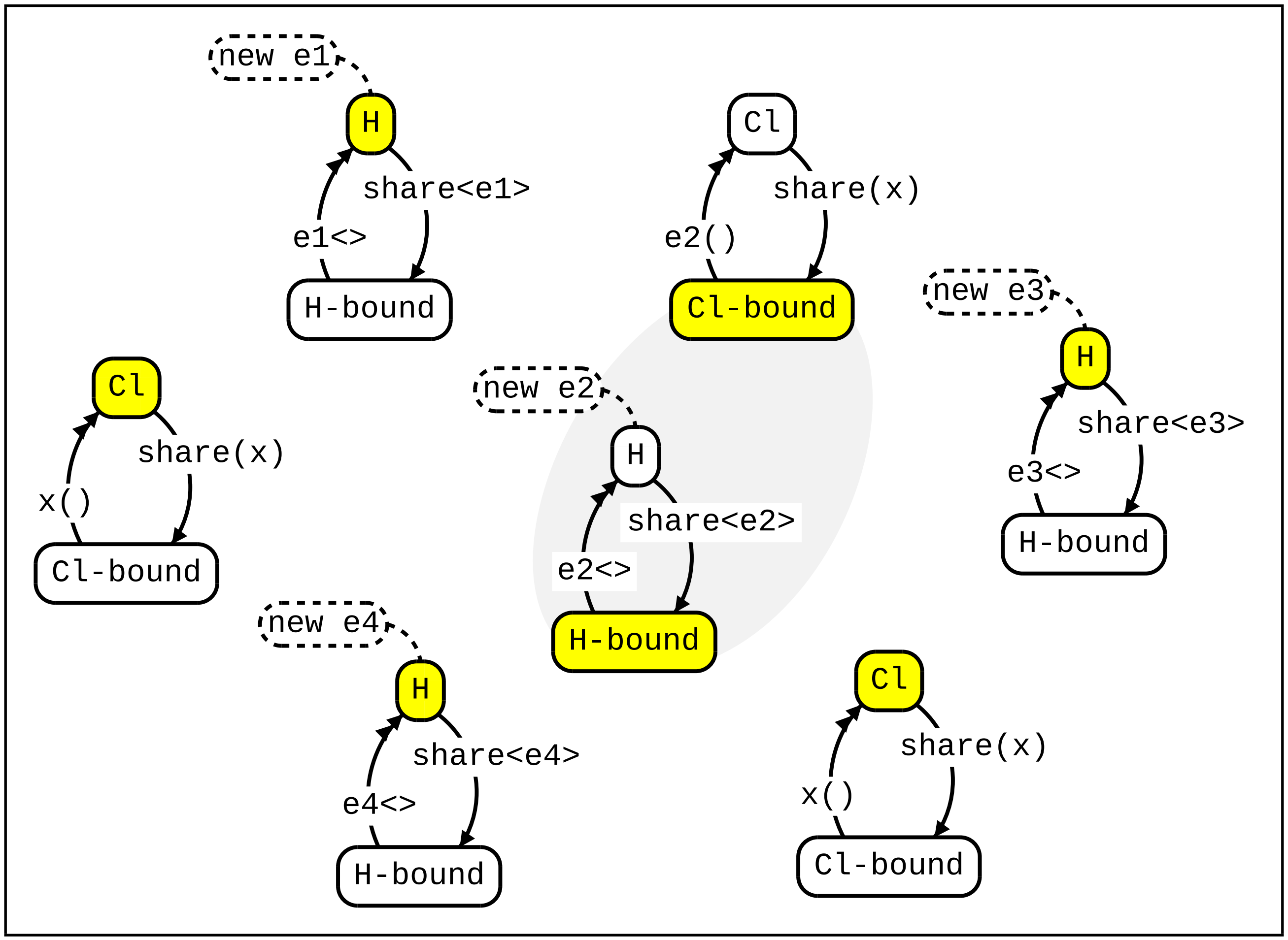
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



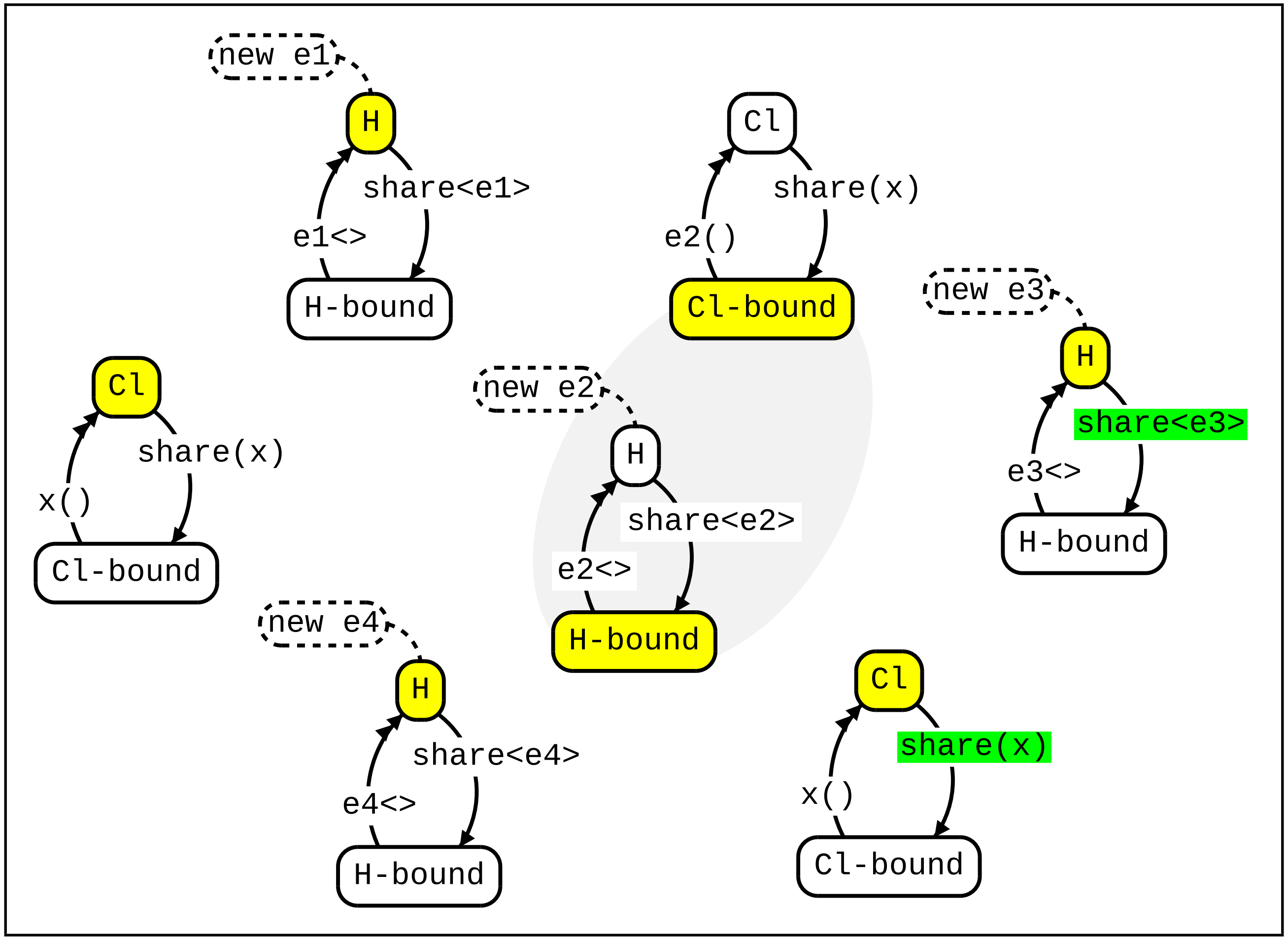
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



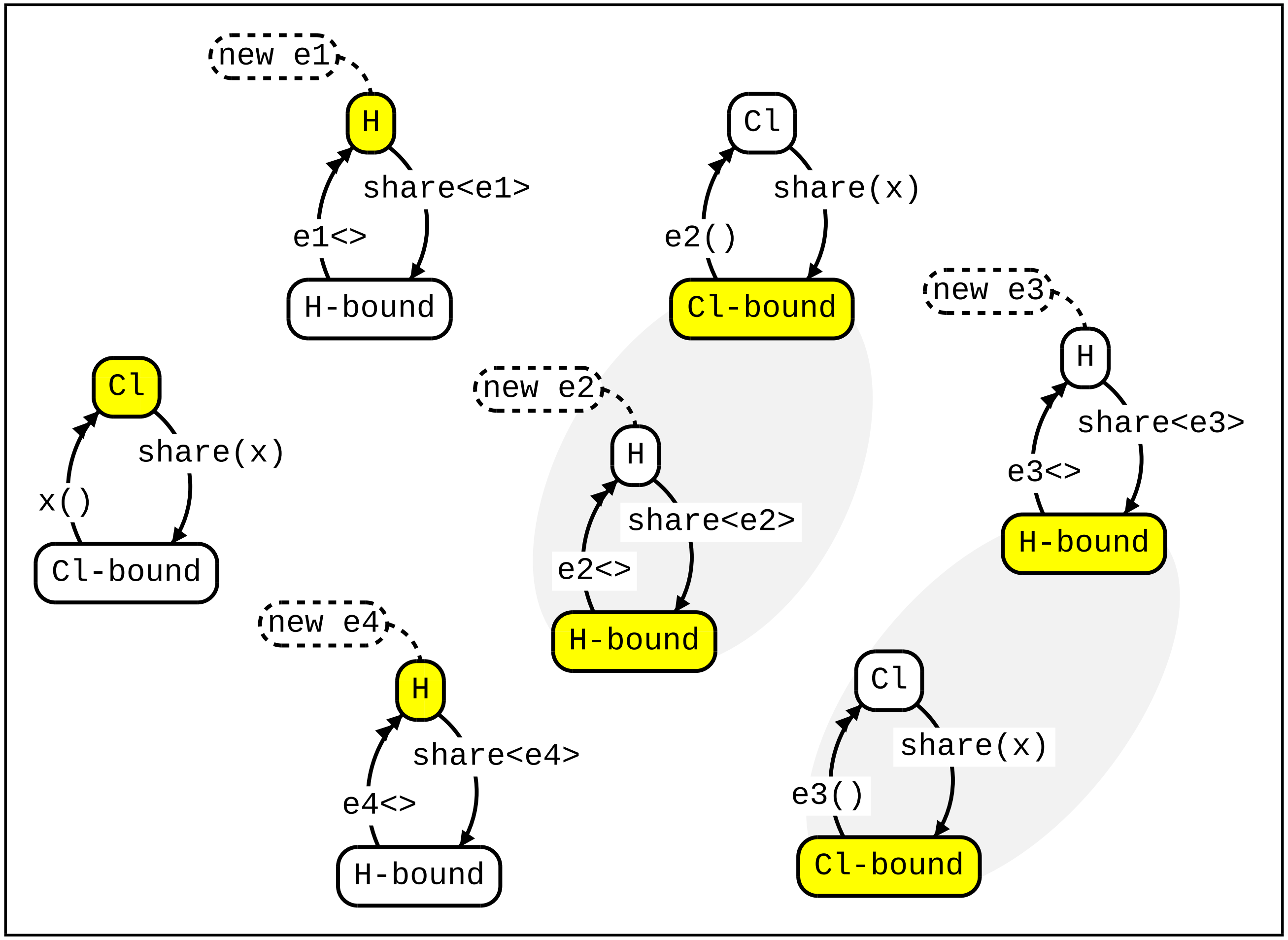
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



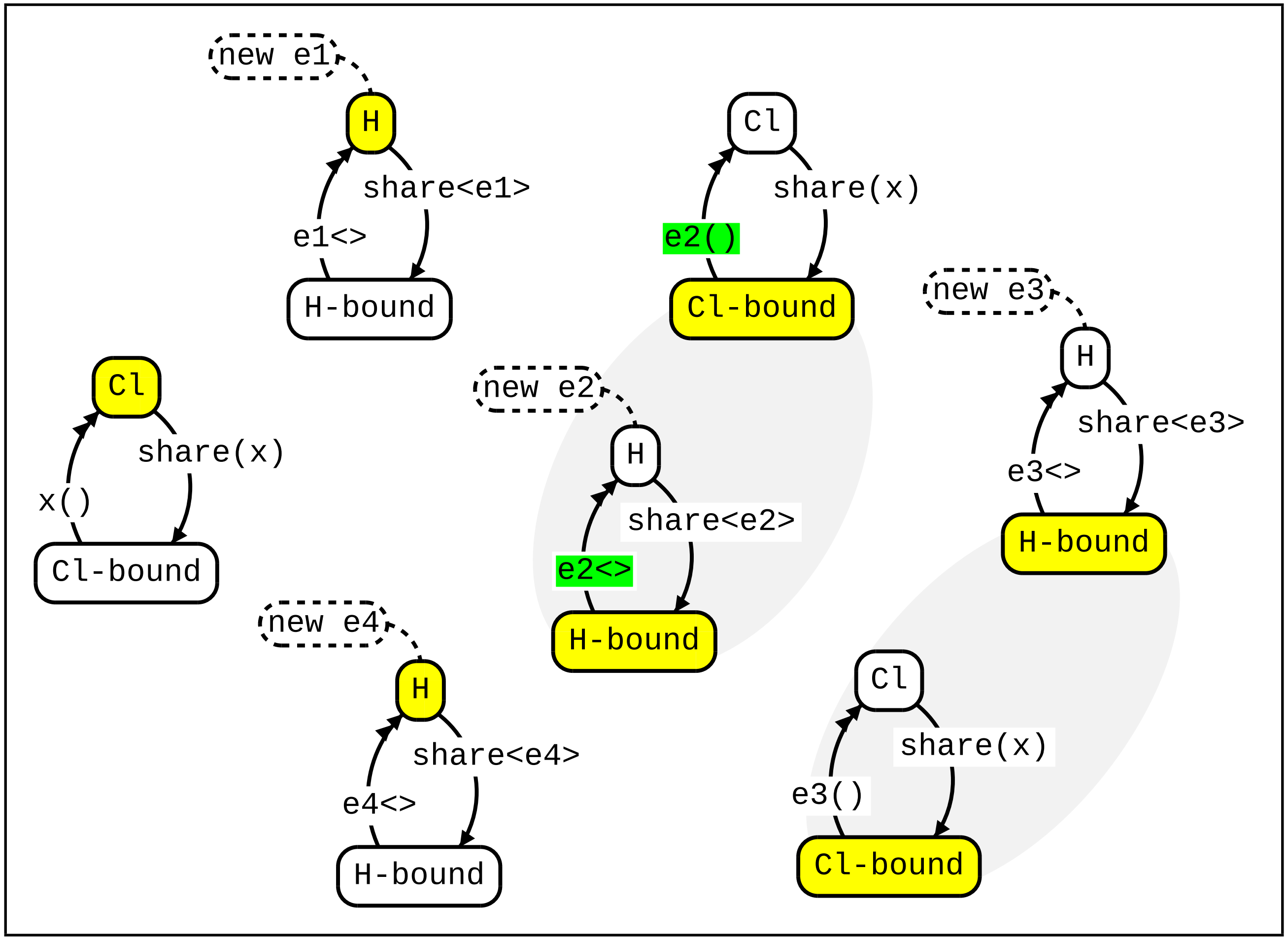
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



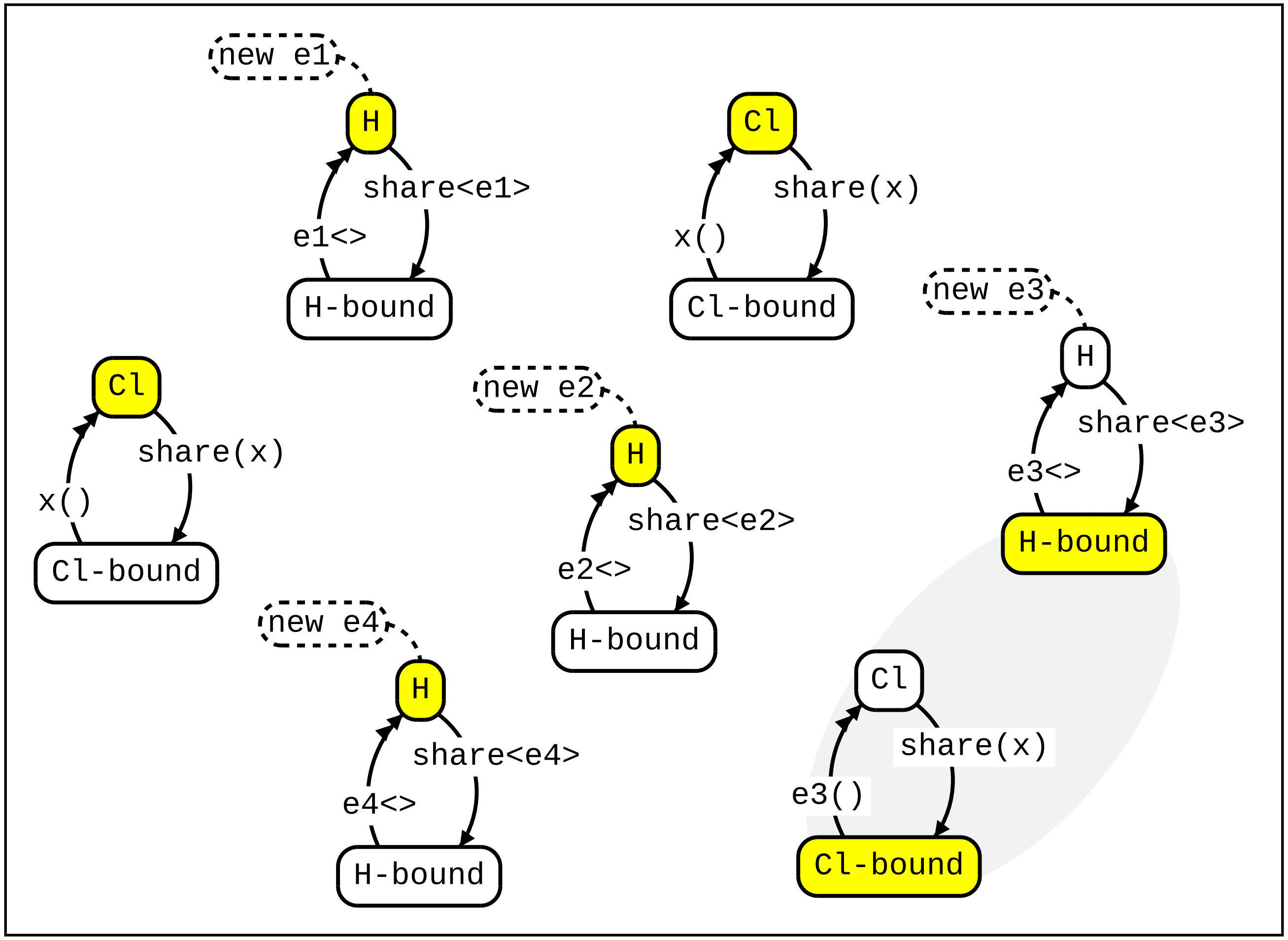
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



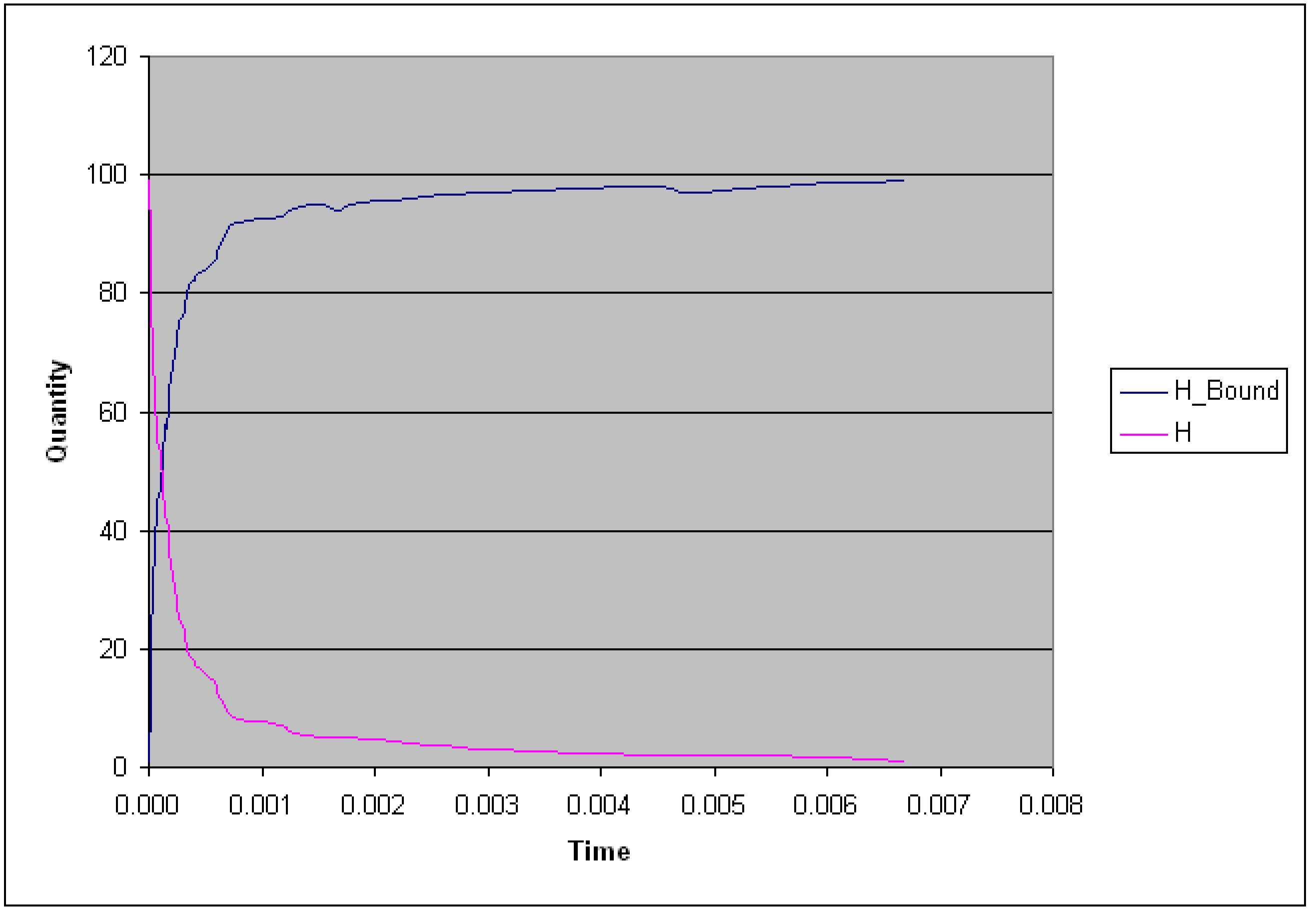
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



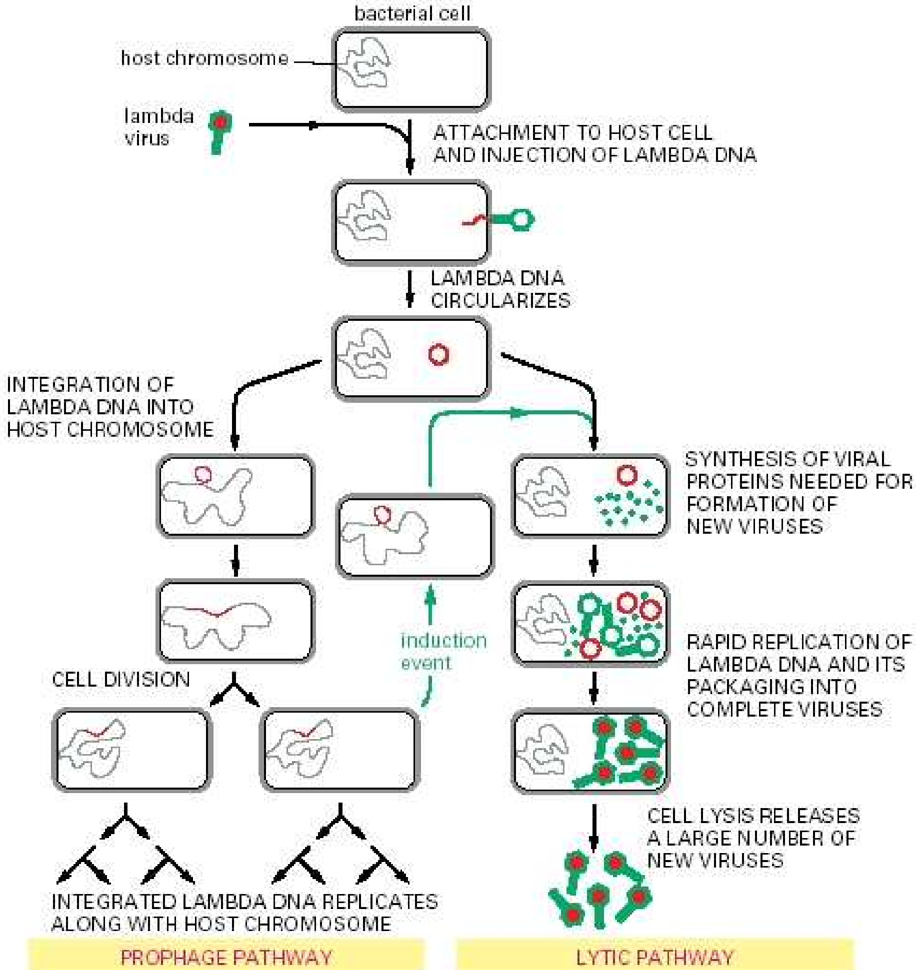
Covalent Bonding: $H + Cl \rightleftharpoons HCl$



Virtual Experiment: $H + Cl \rightleftharpoons HCl$

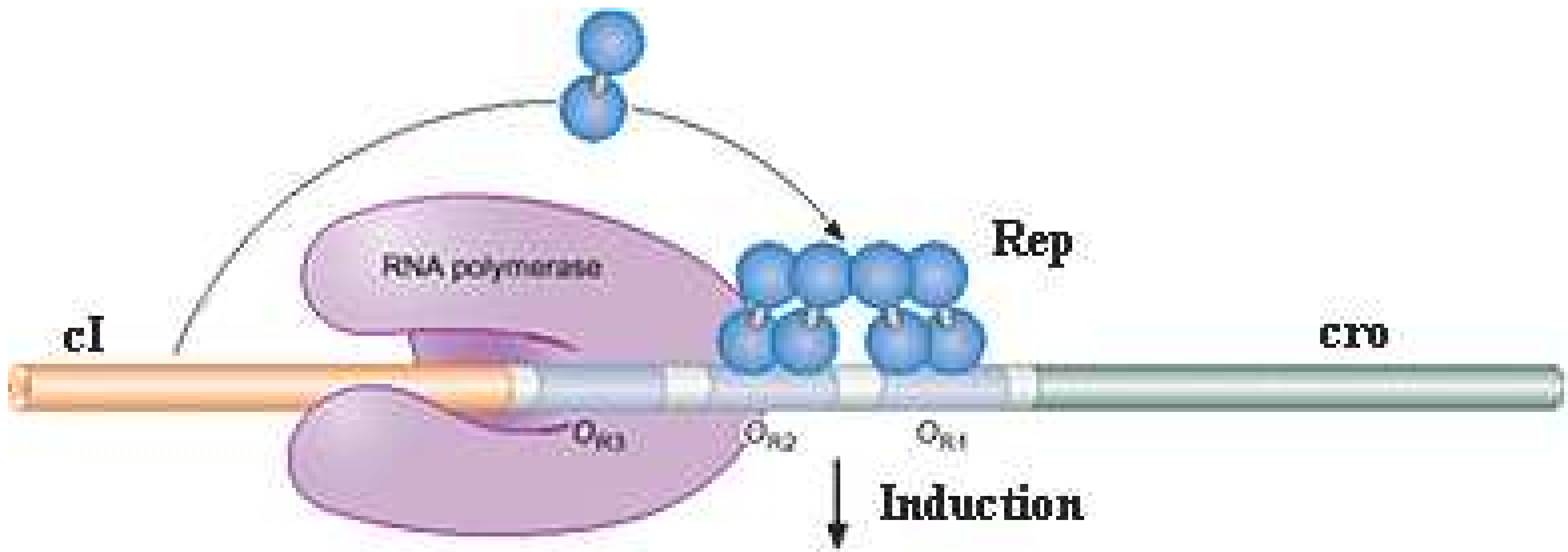


Life Cycle of the Lambda Virus



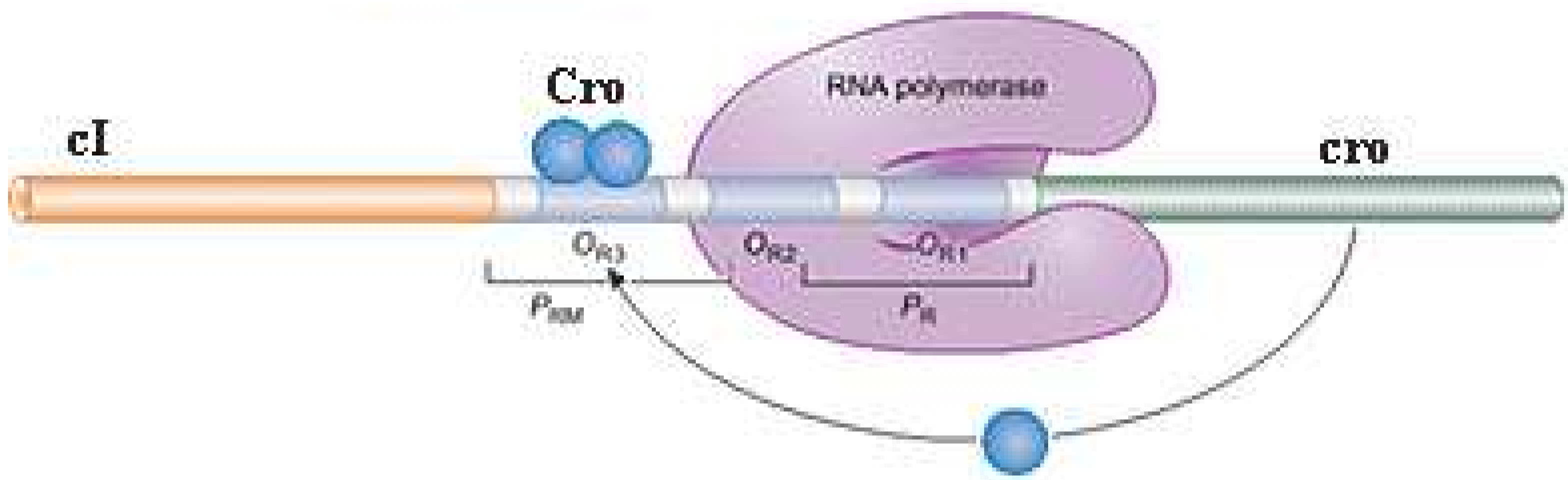
[MBC]

Gene Regulation: Dormant Virus



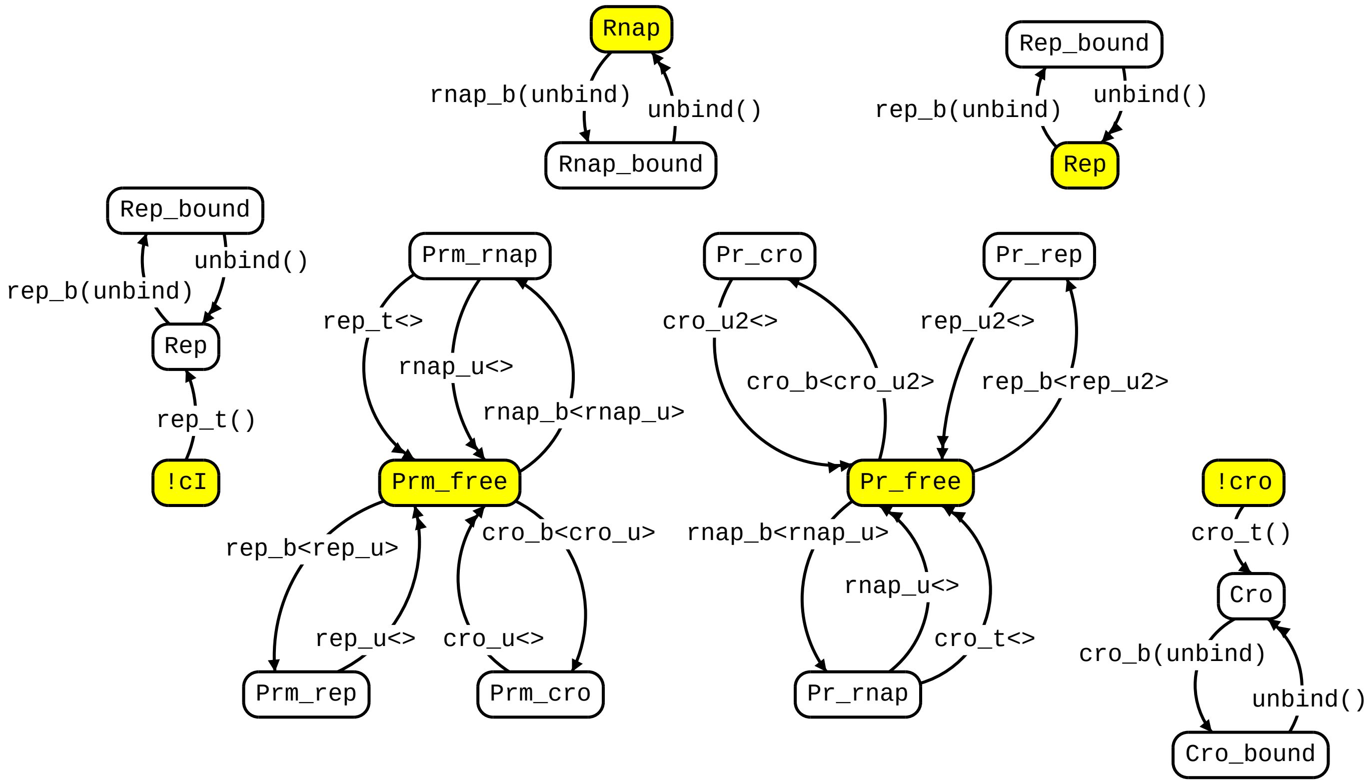
[BLC]

Gene Regulation: Active Virus

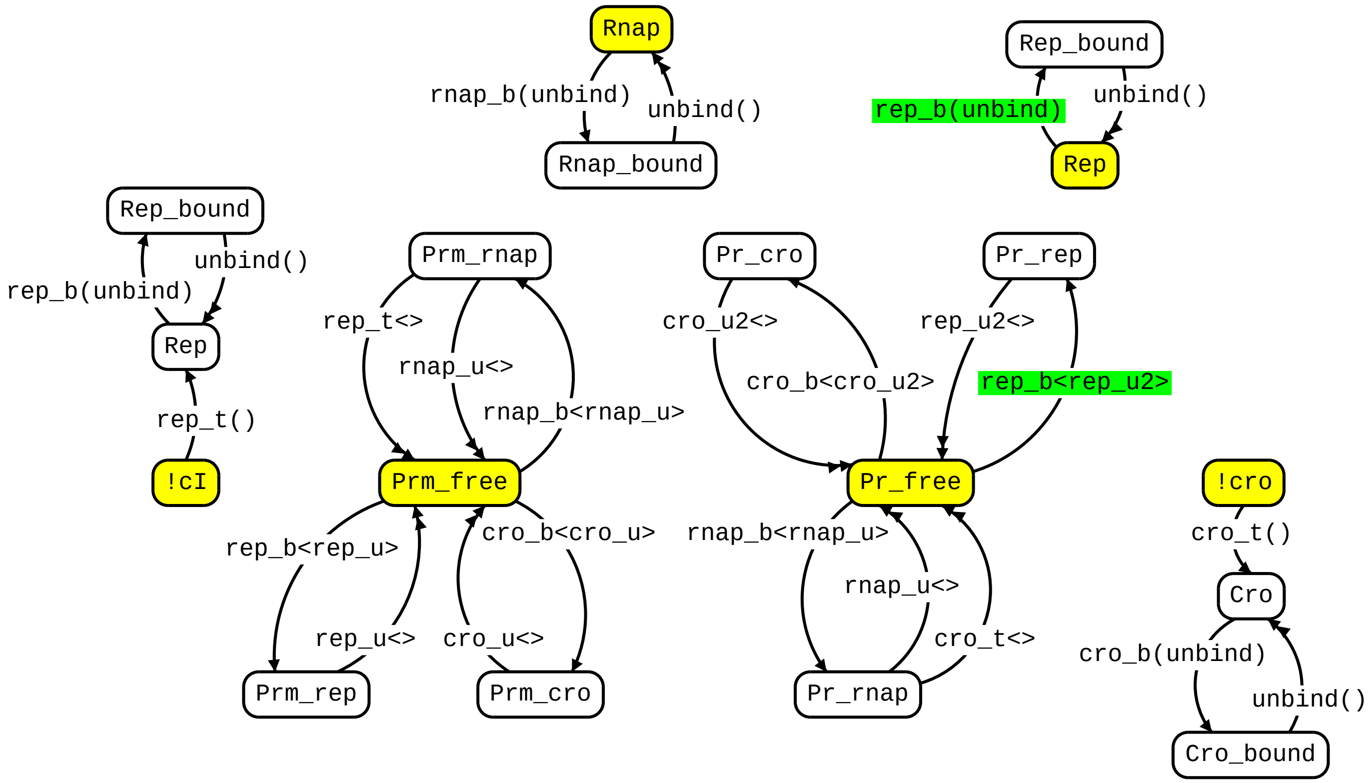


[BLC]

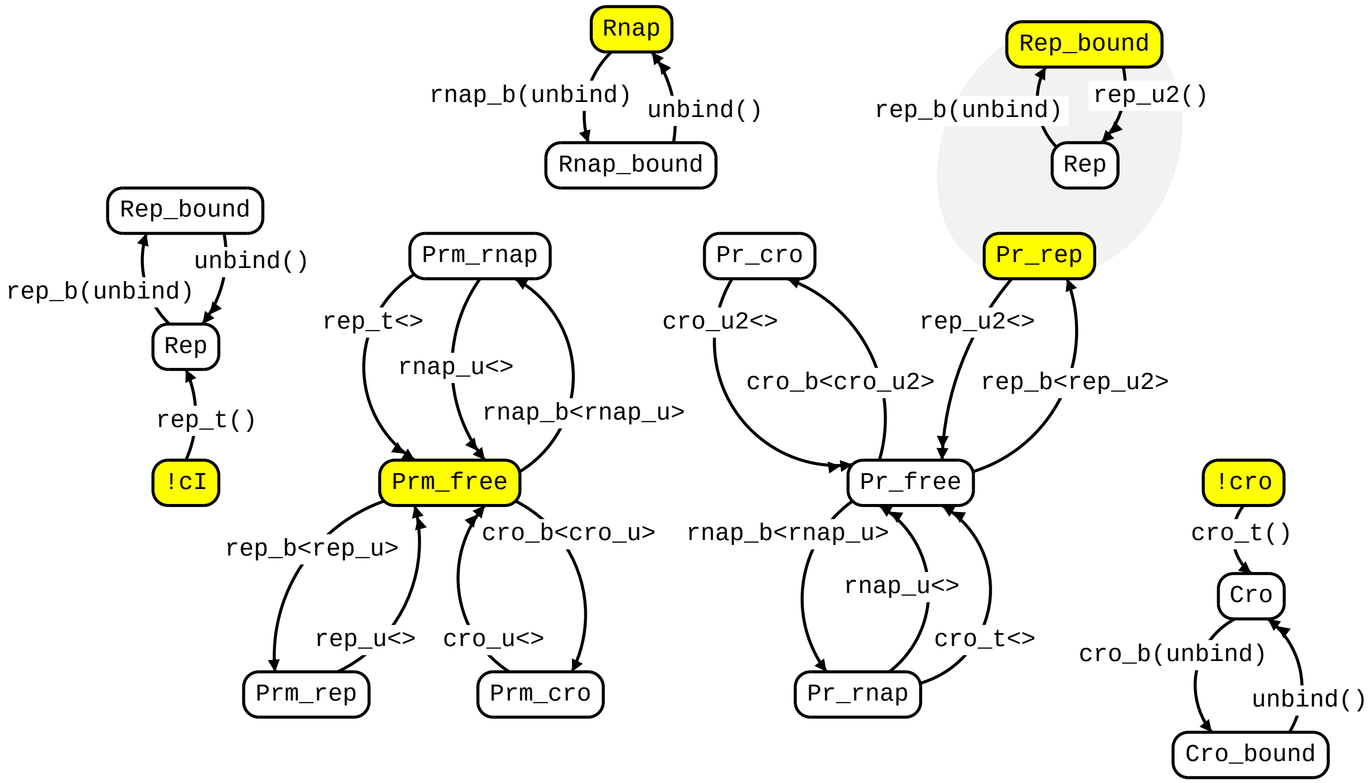
Pi Model: Dormant Virus



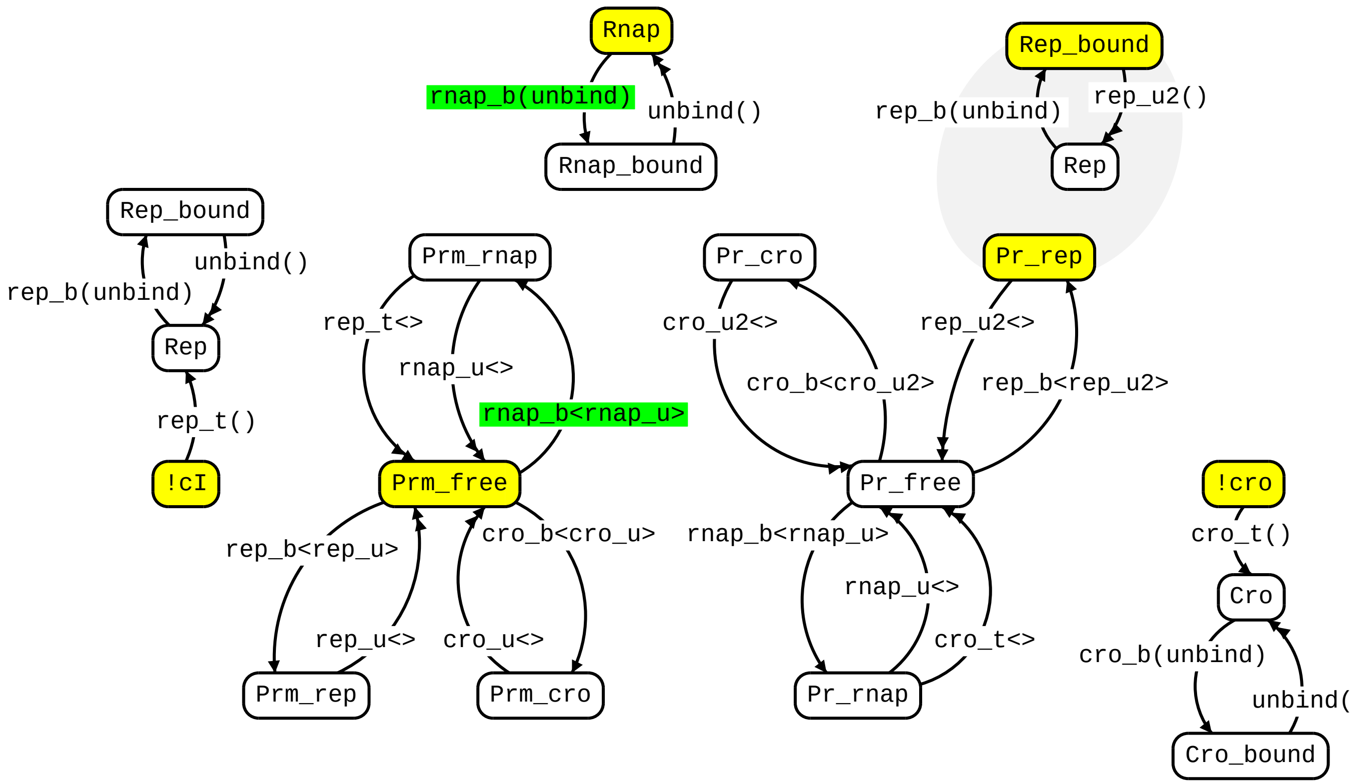
Pi Model: Dormant Virus



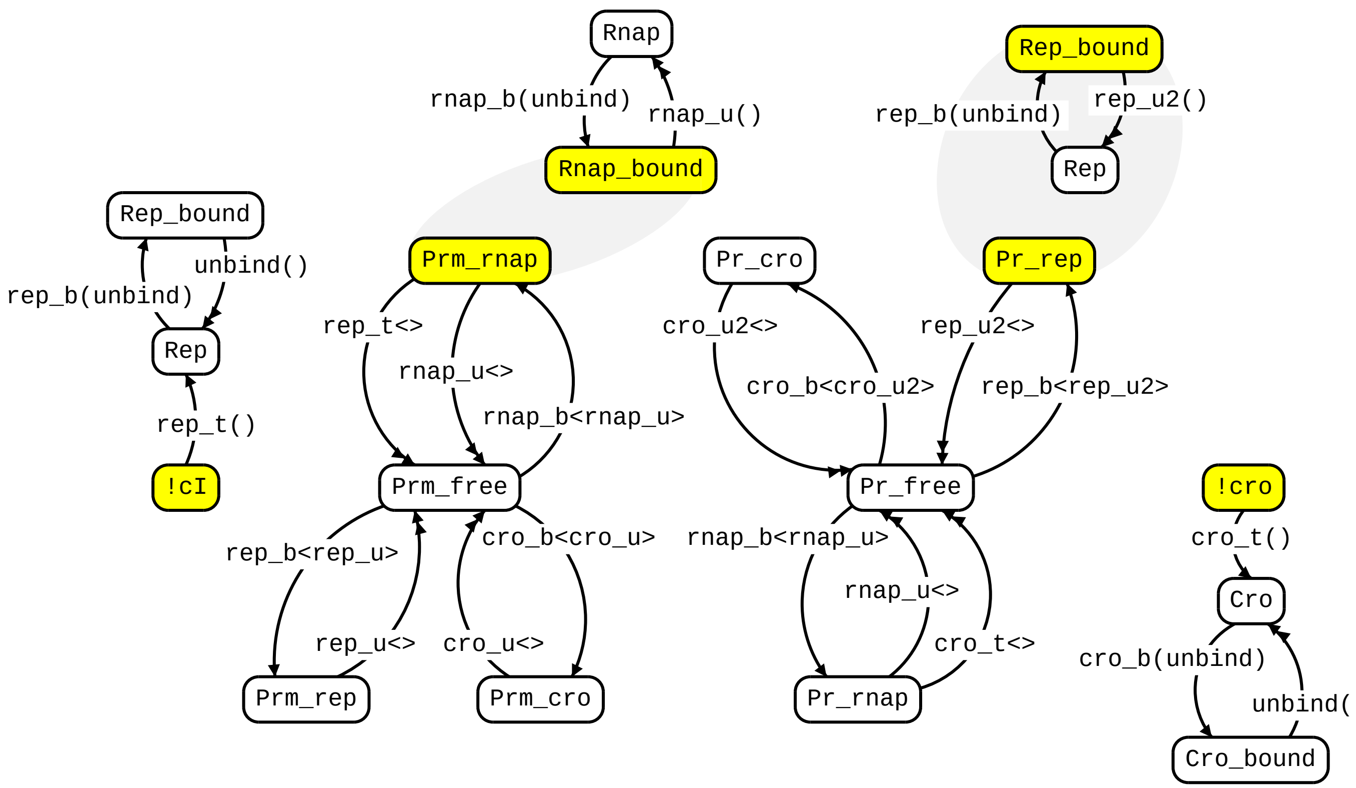
Pi Model: Dormant Virus



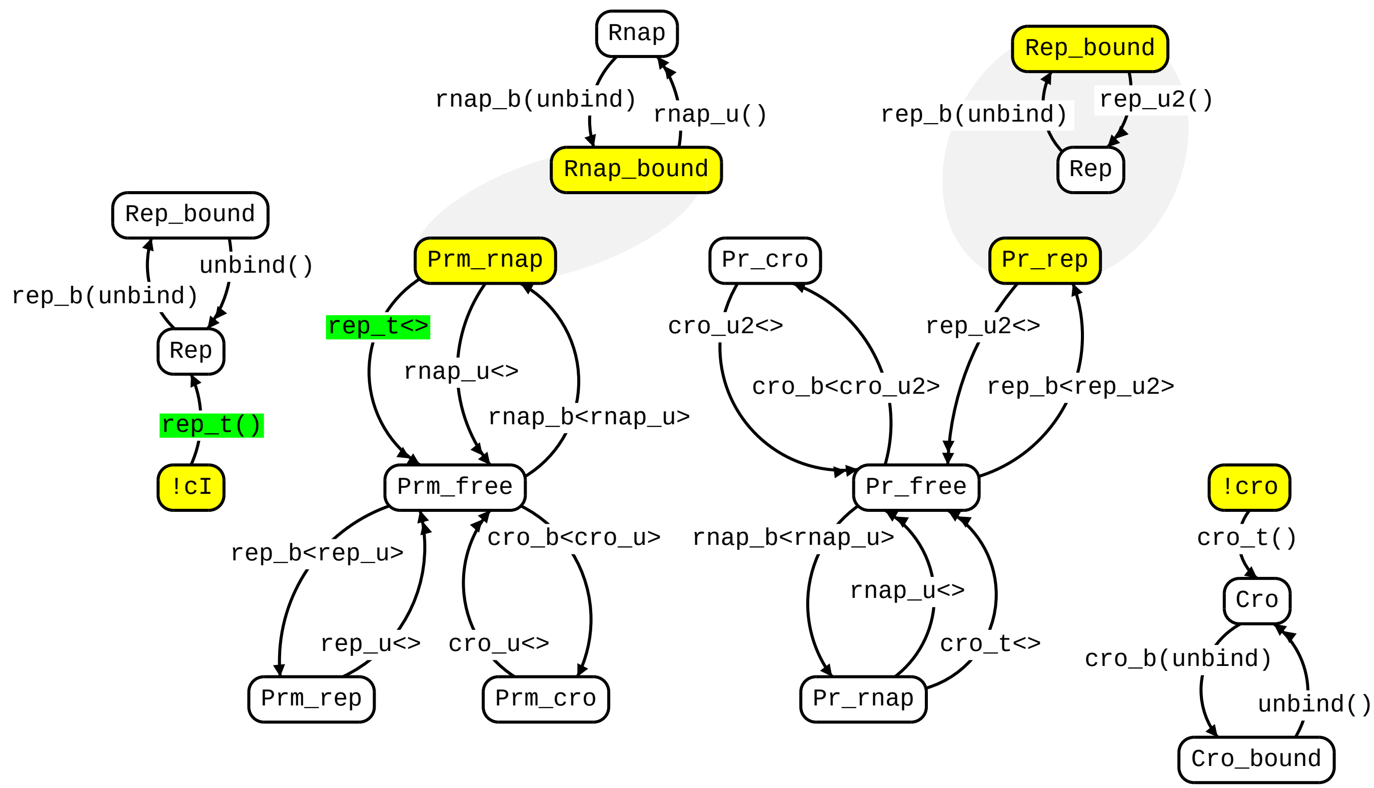
Pi Model: Dormant Virus



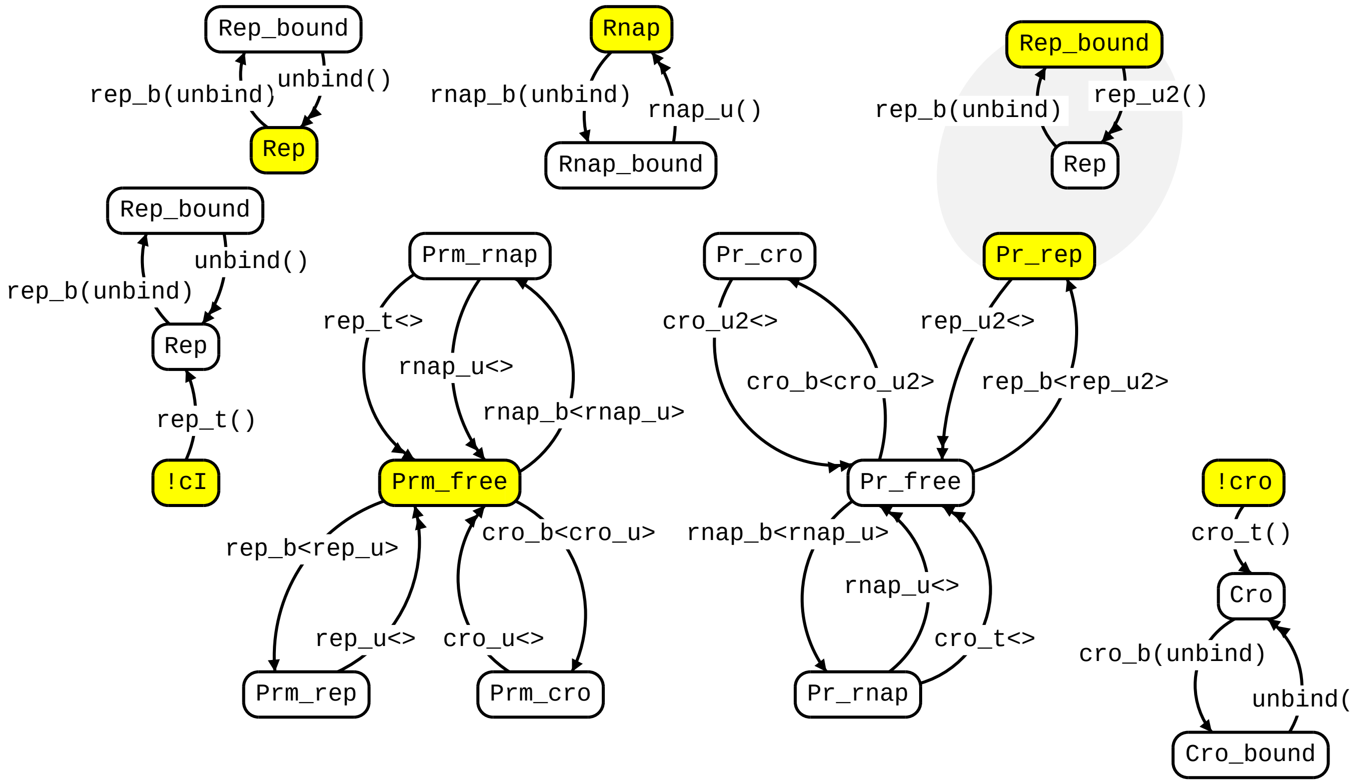
Pi Model: Dormant Virus



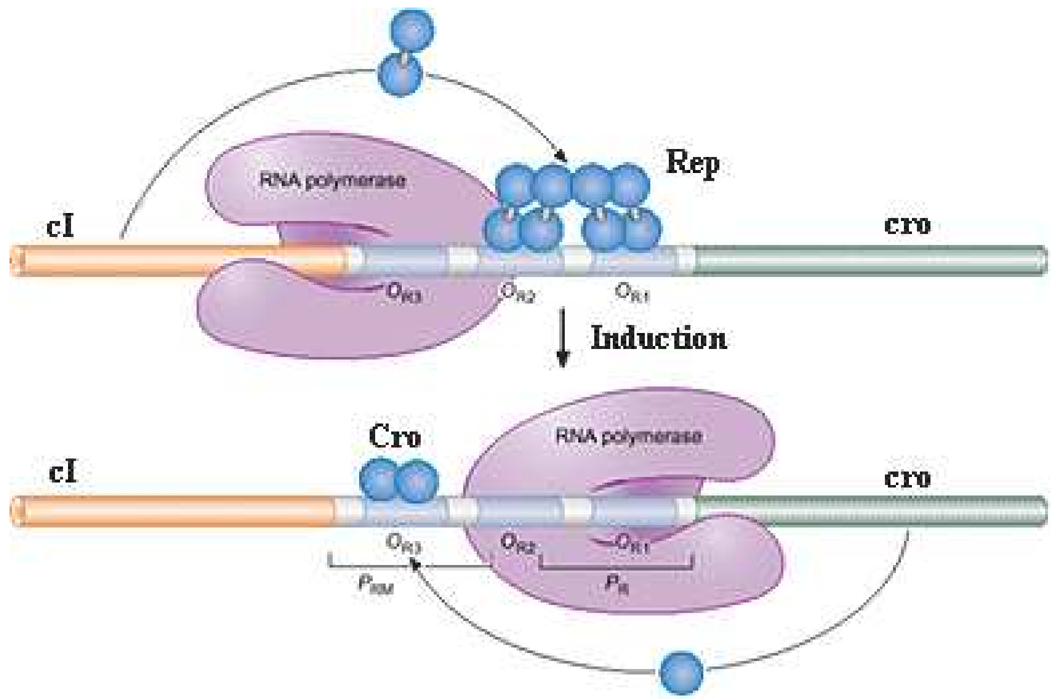
Pi Model: Dormant Virus



Pi Model: Dormant Virus

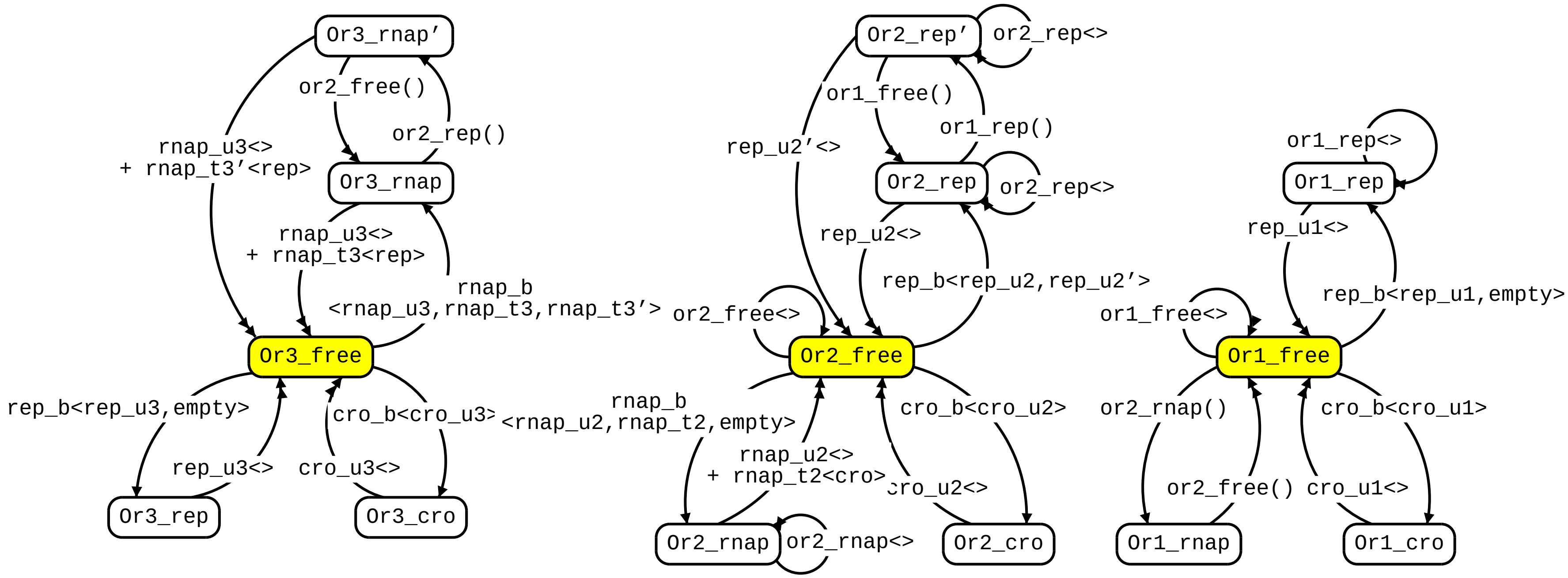


Gene Regulation: Co-operative effects



[BLC]

Pi Model: Co-operative effects



Abstract Machine

- Formalise how the simulator works (program specification).
- Prove properties about the simulator.
- Give greater confidence in the simulation results.
- Improve on existing simulators.

Machine Data Structures

➢ Machine syntax

$$\nu n_1\, \nu n_2 \ldots \nu n_N\, (\Sigma_1 :: \Sigma_2 :: \ldots :: \Sigma_M :: [])$$

:

$$\begin{array}{rcll} V, U & ::= & \nu n\, V & \text{Restriction} \\ & \mid & A & \text{List} \\ A, B & ::= & [] & \text{Empty} \\ & \mid & \Sigma :: A & \text{Summation} \end{array}$$

Machine Encoding

- Encoding $[\![P]\!]$:

$$[\![P]\!] \triangleq P \circ []$$

- Construction $(P \circ V)$:

$$\begin{array}{rcrcl}
n \notin fn(P) & \Rightarrow & P \circ (\nu n\, V) & \triangleq & \nu n\,(P \circ V) \\
 & & \mathbf{0} \circ A & \triangleq & A \\
 & & (P \mid Q) \circ A & \triangleq & P \circ Q \circ A \\
n \notin fn(P \circ A) & \Rightarrow & (\nu m\, P) \circ A & \triangleq & \nu n\,(P_{\{n/m\}} \circ A) \\
 & & !\pi.P \circ A & \triangleq & (\pi.(P \mid !\pi.P) + \mathbf{0}) \circ A \\
 & & (\pi.P + \Sigma) \circ A & \triangleq & (\pi.P + \Sigma) :: A
\end{array}$$

Machine Execution

- Reduction ($V \longrightarrow V'$):

$$V \longrightarrow V' \quad \Rightarrow \quad \nu n\, V \quad \longrightarrow \quad \nu n\, V'$$

$$\left|\begin{array}{l} A \succ (x(m).P + \Sigma) :: A' \\ \wedge A' \succ (x\langle n\rangle.Q + \Sigma') :: A'' \end{array}\right. \quad \Rightarrow \quad A \quad \longrightarrow \quad P_{\{n/m\}} \circ Q \circ A''$$

- Selection:

$$A \quad \succ \quad A$$

$$A \succ \Sigma' :: A' \Rightarrow \Sigma :: A \quad \succ \quad \Sigma' :: \Sigma :: A'$$

$$\Sigma :: A \succ (\pi'.P' + \Sigma') :: A \Rightarrow (\pi.P + \Sigma) :: A \quad \succ \quad (\pi'.P' + \pi.P + \Sigma') :: A$$

Stochastic Machine

- Machine can be easily extended with rates:

$$V \xrightarrow{r} V' \quad \Rightarrow \quad \nu n^{r'} V \quad \xrightarrow{r} \quad \nu n^{r'} V'$$

$$\begin{array}{|l} x^r = Next(A) \\ \wedge A \succ (x^r(m).P + \Sigma) :: A' \\ \wedge A' \succ (x^r\langle n \rangle.Q + \Sigma') :: A'' \end{array} \quad \Rightarrow \quad A \quad \xrightarrow{r} \quad P_{\{n/m\}} \circ Q \circ A''$$

- Choose next reaction $Next(A)$ using a stochastic algorithm (Gillespie)

Channel Activity

- Activity = number of possible interactions on a given channel:

$$\mathrm{Act}_x(A) = (\mathrm{In}_x(A) * \mathrm{Out}_x(A)) - \mathrm{Mix}_x(A)$$

- $\mathrm{In}_x(A)$ = the number of unguarded *inputs* on channel x in A.
- $\mathrm{Out}_x(A)$ = the number of unguarded *outputs* on channel x in A.
- $\mathrm{Mix}_x(A)$ = the sum of $\mathrm{In}_x(\Sigma_i) \times \mathrm{Out}_x(\Sigma_i)$ for each summation Σ_i in A.

Gillespie: Choosing the Next Reaction $Next(A)$

1. For all $x \in fn(A)$ calculate $a_{x^r} = \mathrm{Act}_{x^r}(A) * r$

2. Store non-zero values of a_{x^r} in a list (x_μ, a_μ), where $\mu \in 1...M$.

3. Calculate $a_0 = \sum_{\nu=0}^{M} a_\nu$

4. Randomly generate n_1 and $n_2 \in [0,1]$ and calculate τ and μ such that:

$$\tau = (1/a_0)\ln(1/n_1)$$

$$\sum_{\nu=1}^{\mu-1} a_\nu < n_2 a_0 \leq \sum_{\nu=1}^{\mu} a_\nu$$

5. $Next(A) = x_\mu$ and $Delay(A) = \tau$.

Correctness of the Machine

➢ Safety: no runtime errors (no crashes)

Lemma 1. $\forall V. V \in \mathrm{PiM} \wedge V \longrightarrow V' \Rightarrow V' \in \mathrm{PiM}$

➢ Soundness: machine only performs valid executions steps (behaves well)

Theorem 1. $\forall V. V \in \mathrm{PiM} \wedge V \longrightarrow V' \Rightarrow [\![V]\!] \longrightarrow [\![V']\!]$

➢ Completeness: machine accurately executes all behaviours of the calculus

Theorem 2. $\forall P. P \in \mathrm{Pi} \wedge P \longrightarrow P' \Rightarrow (\!|P|\!) \longrightarrow^{\equiv} (\!|P'|\!)$.

➢ Termination: machine does not loop forever unnecessarily

Theorem 3. $\forall P. P \in \mathrm{Pi} \wedge P \not\longrightarrow \Rightarrow (\!|P|\!) \not\longrightarrow$

Stochastic Correctness

- Theorems easily extend to reductions with rates ($\xrightarrow{r}$)
- Need to take into account the number of possible interactions on a channel:

$$
\begin{aligned}
P_1 &\triangleq x^r\langle n\rangle.P + x^r\langle n\rangle.P \mid x^r(m).Q \\
P_2 &\triangleq x^r\langle n\rangle.P \mid x^r(m).Q
\end{aligned}
$$

- Reduction in P_1 is twice as fast as the reduction in P_2
- Ensure that the reactions in the machine have the same rates as in the calculus

Proposition 1. $\forall V \in \mathrm{PiM}.\mathrm{App}_{x^r}(V) = \mathrm{App}_{x^r}([\![V]\!])$

Proposition 2. $\forall P \in \mathrm{Pi}.\mathrm{App}_{x^r}(P) = \mathrm{App}_{x^r}((\!|P|\!))$

Implementation

- Abstract Machine maps almost directly to program code
- Implemented a polymorphic type system and type checker
- Correctness of the machine gives greater confidence in the simulation results

Conclusion

- Presented a graphical representation for pi-calculus:
  - Precise, compositional, executable descriptions.
  - Used to model regulatory systems at the molecular level.

- Presented an abstract machine for the stochastic pi-calculus:
  - Proof of correctness (safety, soundness, completeness, termination).
  - Maps readily to program code.

- Built a simulator based on the machine.

Safety Proof

Lemma 2. $\forall V. V \in \mathrm{PiM} \wedge V \longrightarrow V' \Rightarrow V' \in \mathrm{PiM}$

Proof. By Lemma 3, Lemma 4 and by induction on reduction in PiM. □

Lemma 3. $\forall A \in \mathrm{PiM}. A \succ B \Rightarrow B \in \mathrm{PiM}$

Proof. By induction on selection in PiM. □

Lemma 4. $\forall V. \forall P. V \in \mathrm{PiM} \wedge P \in \mathrm{Pi} \Rightarrow P \circ V \in \mathrm{PiM}$

Proof. By induction on construction in PiM. □

Soundness Proof

Lemma 5. $\forall V. V \in \mathrm{PiM} \Rightarrow [\![V]\!] \in \mathrm{Pi}$

Proof. By induction on decoding in PiM. $\square$

Theorem 4. $\forall V. V \in \mathrm{PiM} \wedge V \longrightarrow V' \Rightarrow [\![V]\!] \longrightarrow [\![V']\!]$

Proof. By Lemma 6, Lemma 7 and by induction on reduction in PiM. $\square$

Lemma 6. $\forall A. A \in \mathrm{PiM} \wedge A \succ B \Rightarrow [\![A]\!] \equiv [\![B]\!]$

Proof. By induction on selection in PiM. $\square$

Lemma 7. $\forall V. \forall P. V \in \mathrm{PiM} \wedge P \in \mathrm{Pi} \Rightarrow [\![P \circ V]\!] \equiv P \mid [\![V]\!]$

Proof. By induction on construction in PiM. $\square$

$$[\![\nu n\, V]\!] \triangleq \nu n\, [\![V]\!] \tag{1}$$

$$[\![[]]\!] \triangleq \mathbf{0} \tag{2}$$

$$[\![\Sigma :: A]\!] \triangleq \Sigma \mid [\![A]\!] \tag{3}$$

Completeness Proof

Lemma 8. $\forall V.V \in \mathrm{PiM} \wedge U \equiv V \wedge V \longrightarrow V' \Rightarrow \exists U'.U \longrightarrow U' \wedge U' \equiv V'$

Proof. By induction on structural congruence in PiM. $\square$

Theorem 5. $\forall P.P \in \mathrm{Pi} \wedge P \longrightarrow P' \Rightarrow (\!|P|\!) \longrightarrow\equiv (\!|P'|\!)$.

Proof. By Lemma 9 and by induction on reduction in Pi, where the rule for parallel composition is expanded over the remaining rules. $\square$

Lemma 9. $P \equiv Q \Rightarrow (\!|P|\!) \equiv (\!|Q|\!)$

Proof. By induction on structural congruence in Pi. $\square$

$$
\begin{aligned}
V \equiv_\alpha U \Rightarrow V &\equiv U \\
n \notin fn(V) \quad \Rightarrow \quad \nu n\, V &\equiv V \\
\nu x\, \nu y\, V &\equiv \nu y\, \nu x\, V
\end{aligned}
$$

$$\Sigma :: \Sigma' :: A \equiv \Sigma' :: \Sigma :: A$$

$$A \equiv A' \quad \Rightarrow \quad \Sigma :: A \equiv \Sigma :: A'$$

$$(\pi.P + \pi'.P' + \Sigma) :: A \equiv (\pi'.P' + \pi.P + \Sigma) :: A$$

$$\Sigma :: A \equiv \Sigma' :: A \Rightarrow (\pi.P + \Sigma) :: A \equiv (\pi.P + \Sigma') :: A$$

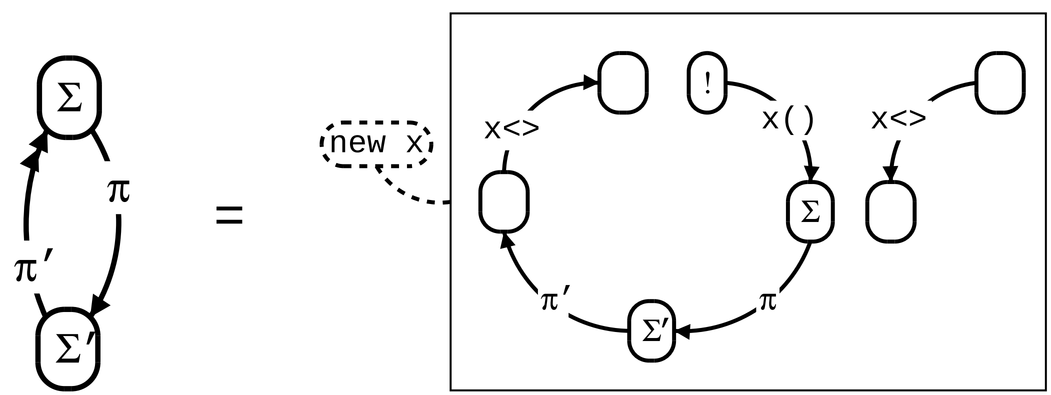
Termination Proof

Theorem 6. $\forall P . P \in \mathrm{Pi} \wedge P \not\longrightarrow \Rightarrow [\![P]\!] \not\longrightarrow$

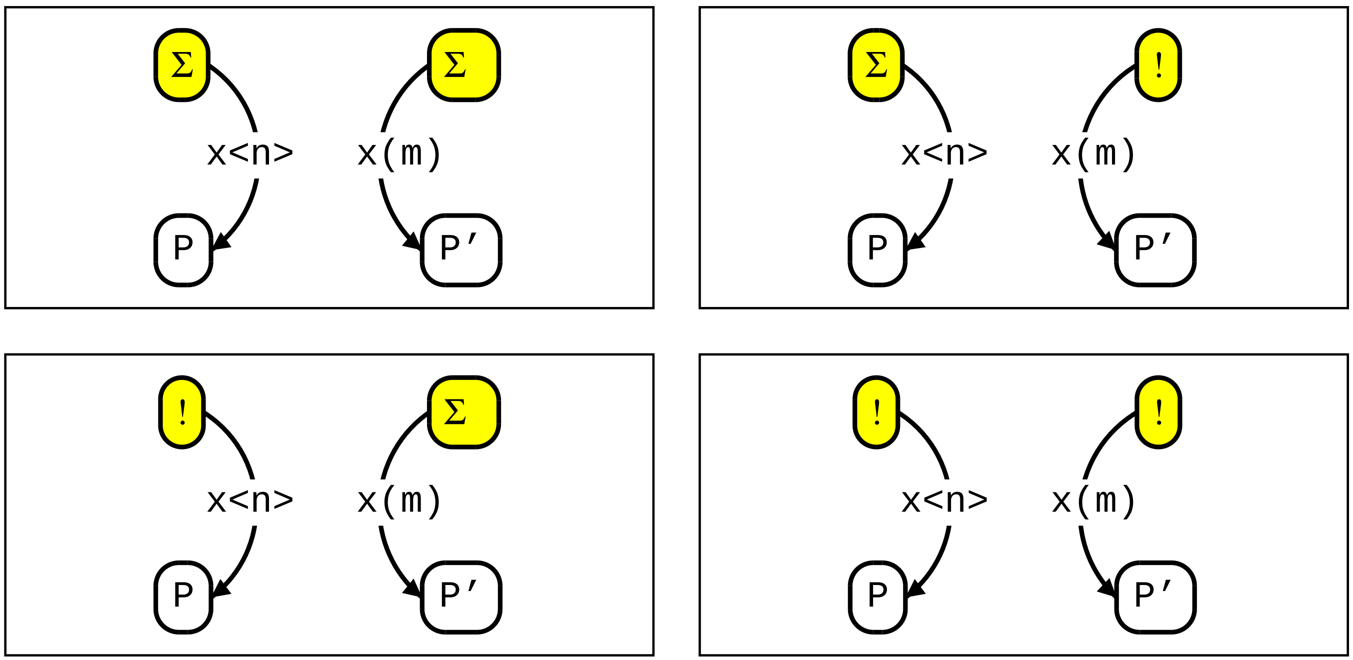
Proof. By Theorem 4 and by basic relationships between encoding and decoding. $\square$

Link Encoding

- Encoding uses restriction, replication, parallel composition and communication.
- A linked node $\longrightarrow$ a replicated input on a fresh channel x, in parallel with an output on x
- A link to the node $\longrightarrow$ an output on x.
- E.g.:

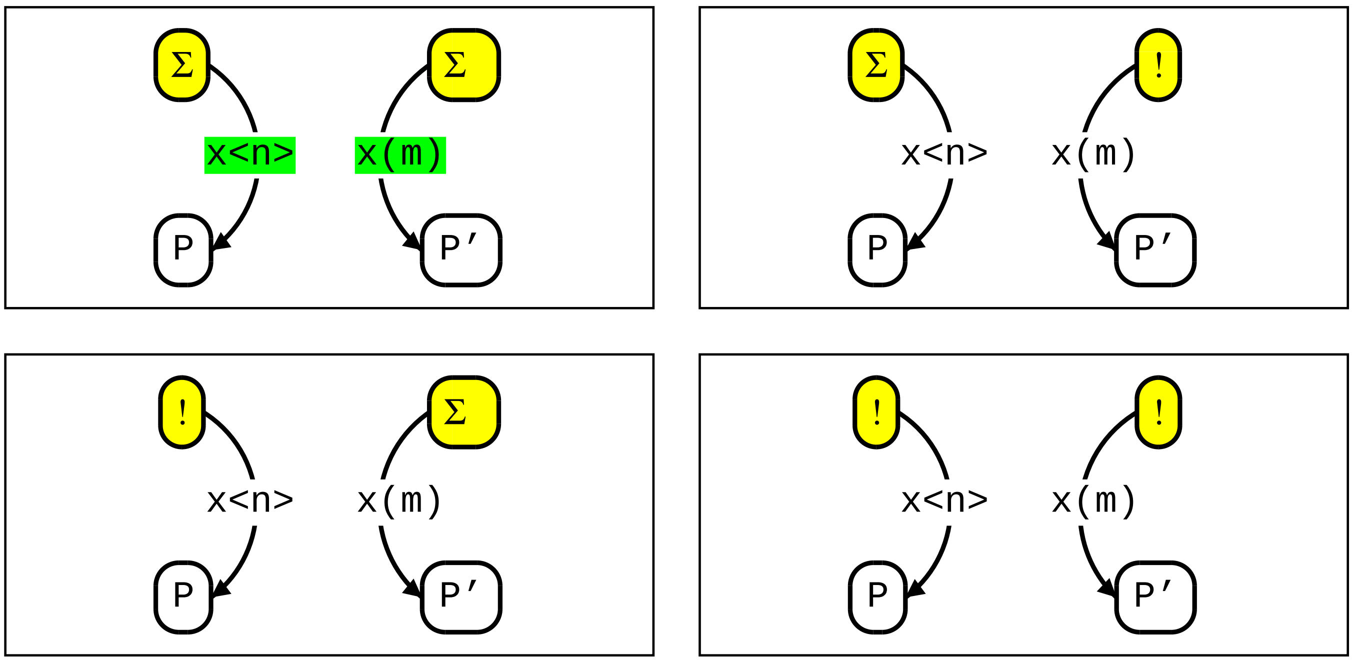


Graphical Semantics



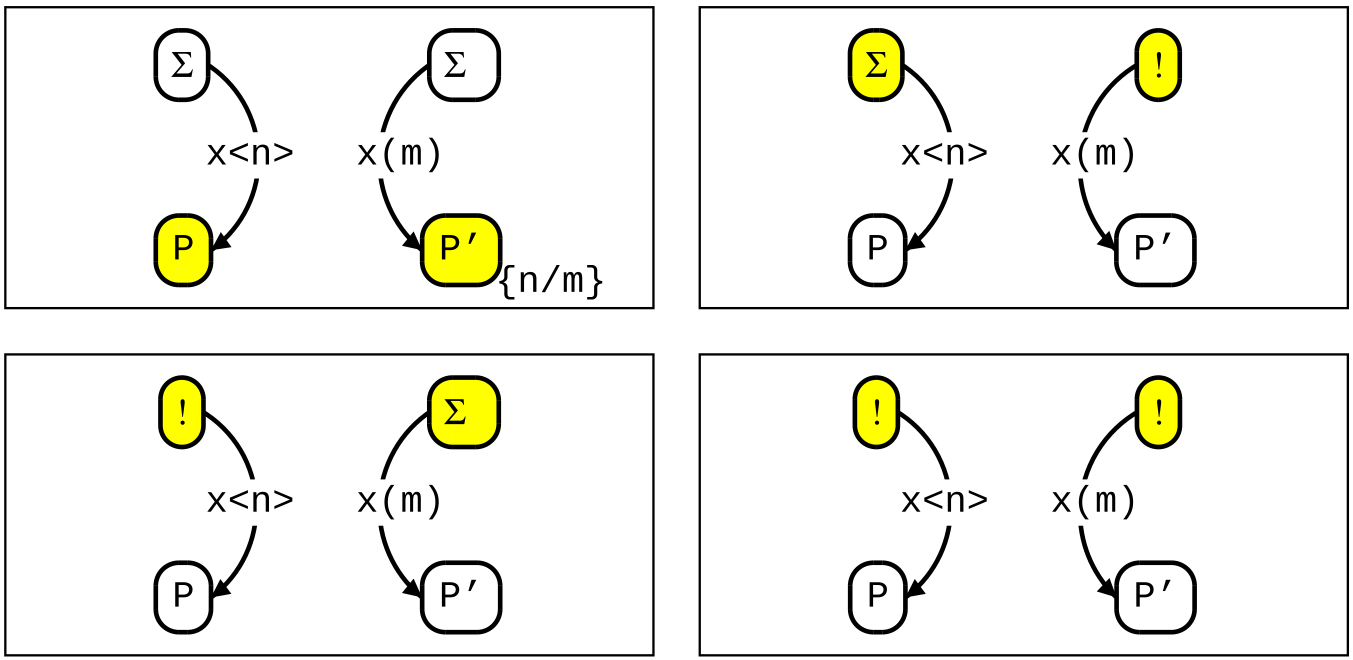
- Requires some imagination: for substituting names and for cloning graphs.

Graphical Semantics



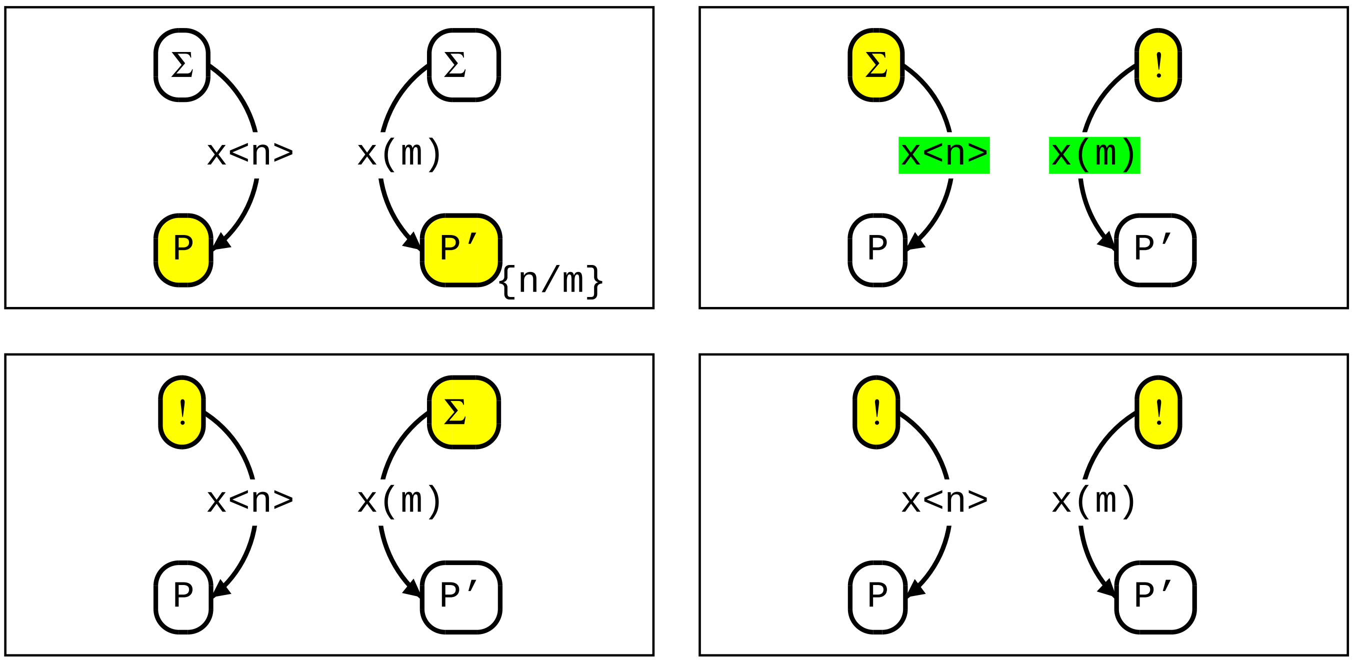
➢ Output $x\langle n\rangle$ can send a message to input $x(m)$ on channel x.

Graphical Semantics



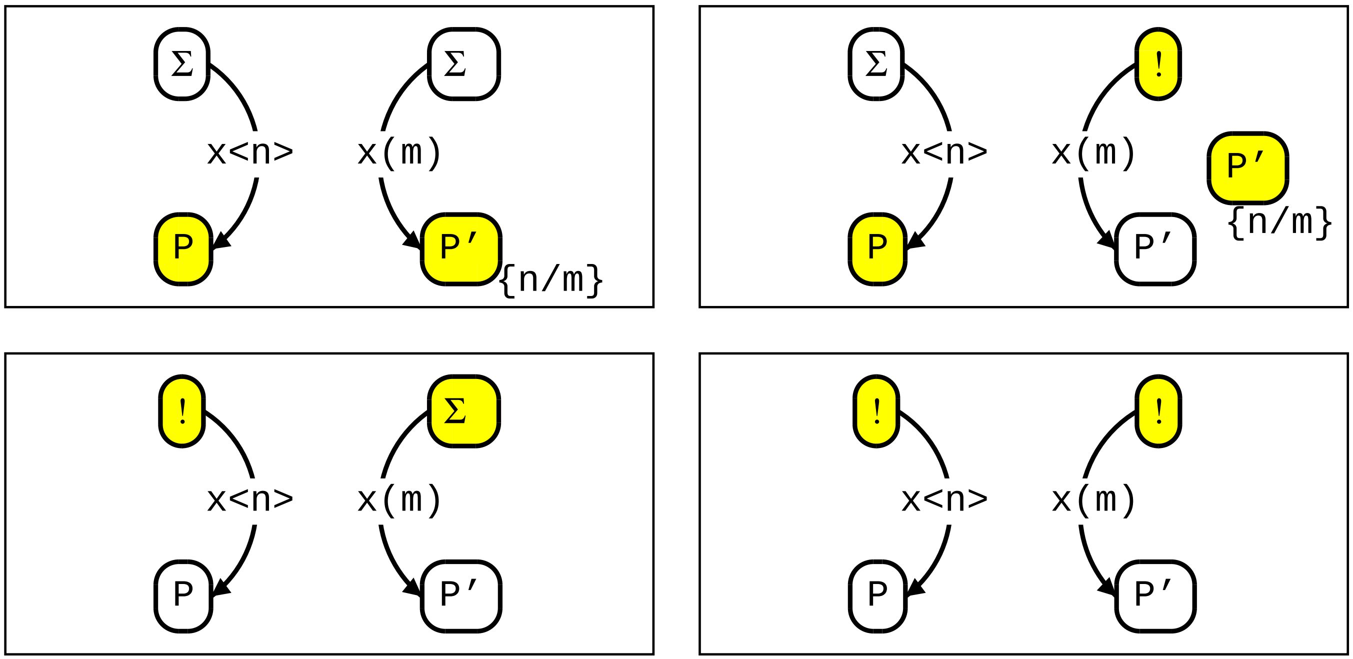
➢ n is assigned to m in process P'.

Graphical Semantics



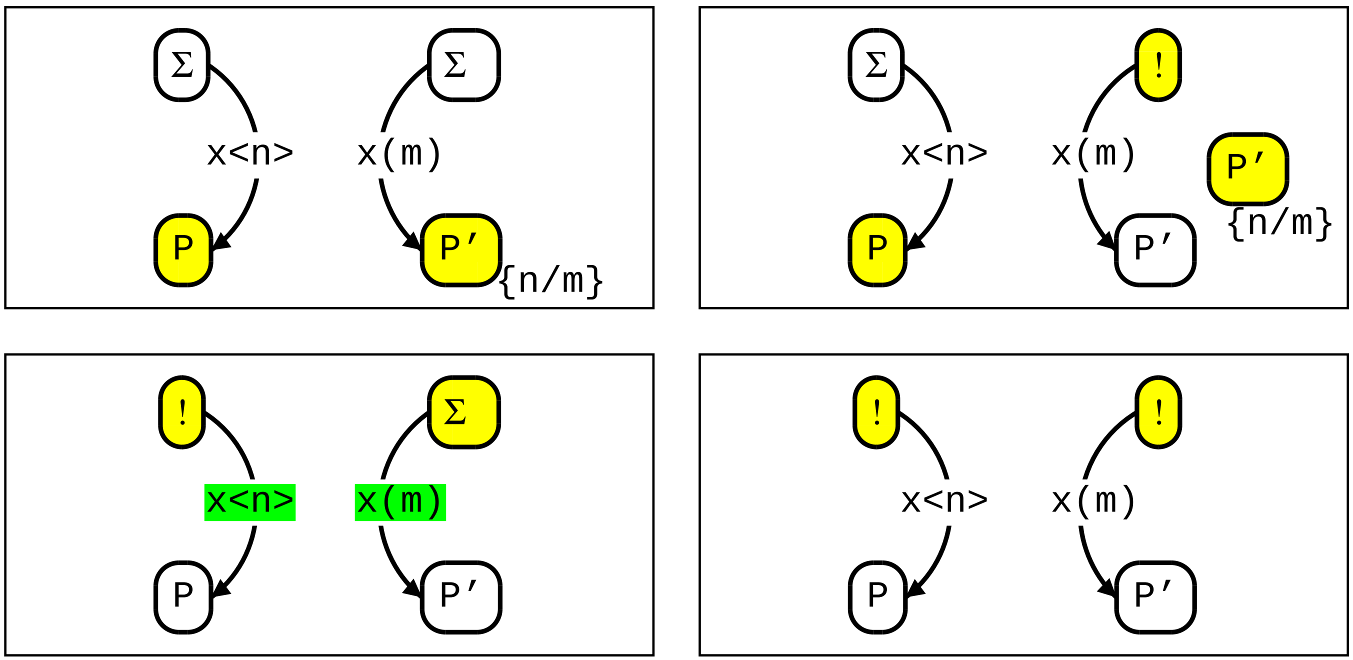
➢ Output $x\langle n\rangle$ can send a message to replicated input $!x(m)$.

Graphical Semantics



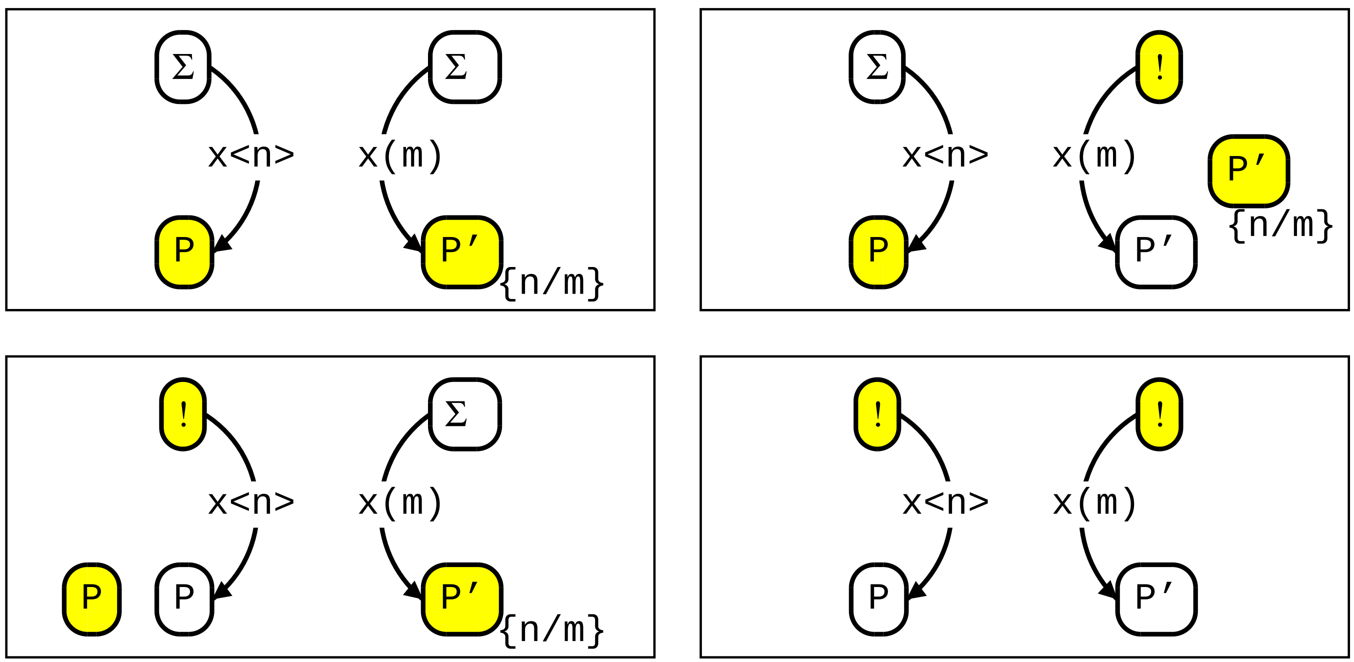
➢ A clone of P' is spawned and n is assigned to m in the clone of P'.

Graphical Semantics



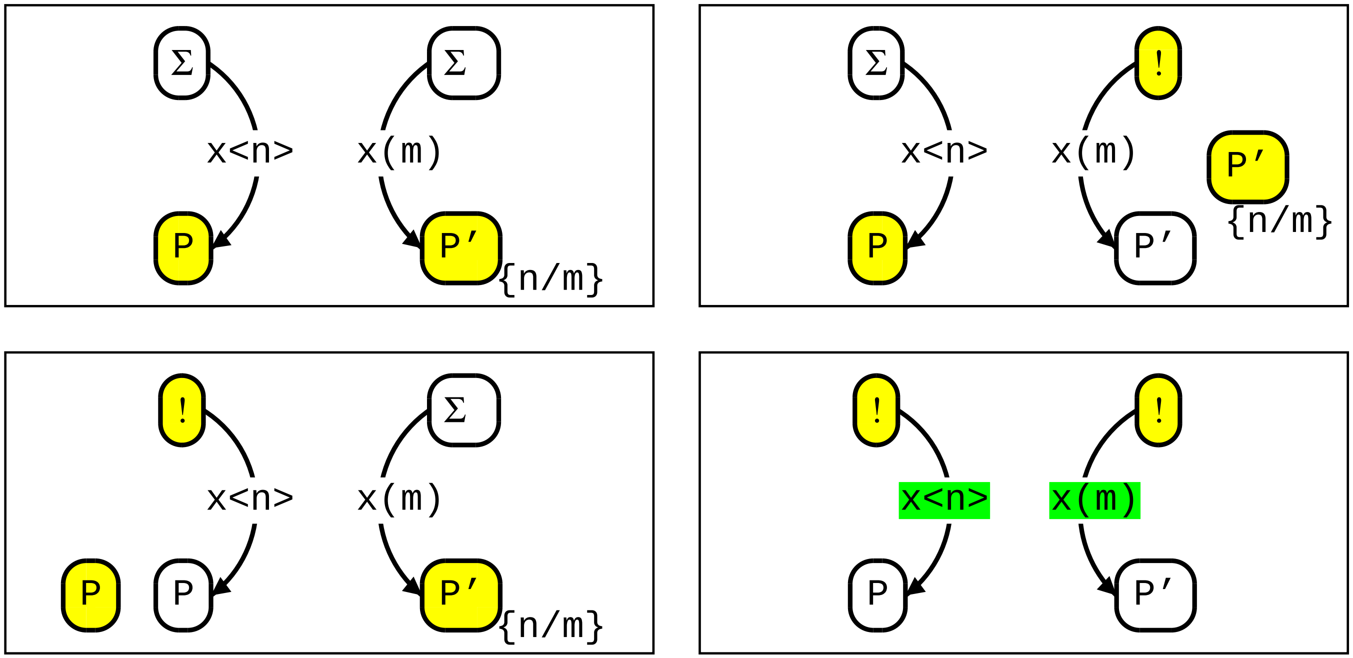
➢ Replicated output $!x\langle n\rangle$ can send a message to input $x(m)$.

Graphical Semantics



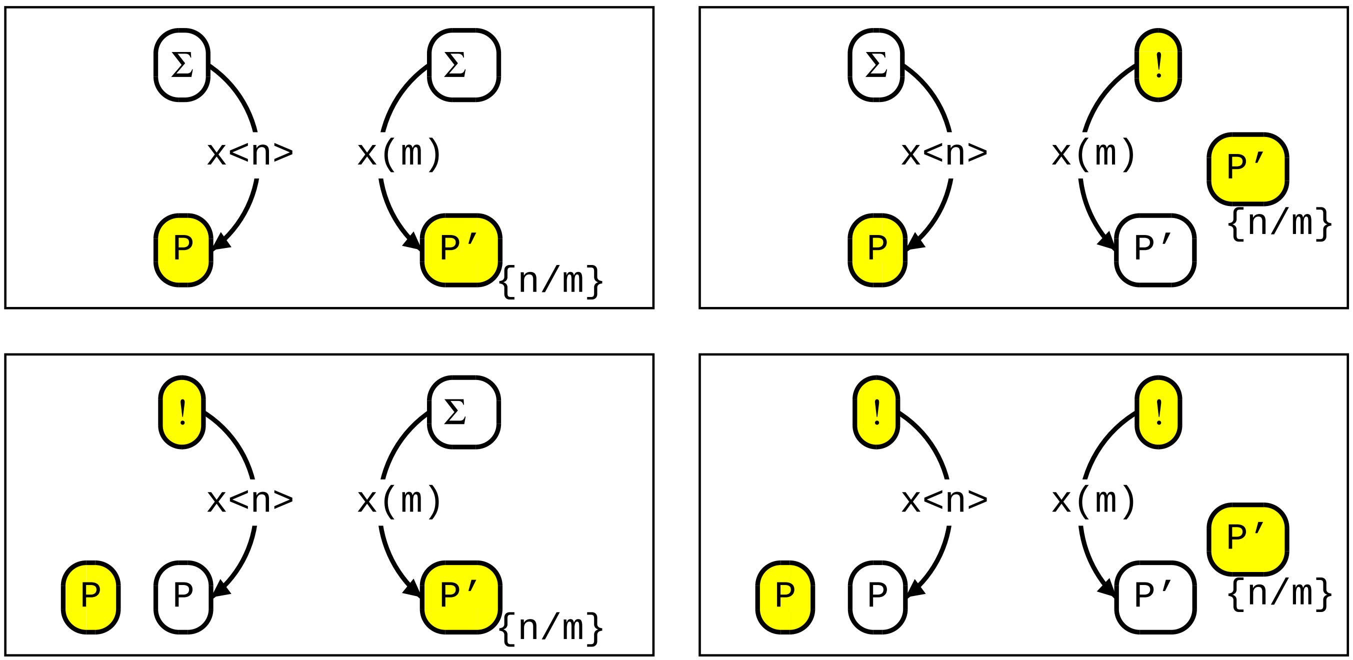
➢ A clone of P is spawned and n is assigned to m in P'.

Graphical Semantics



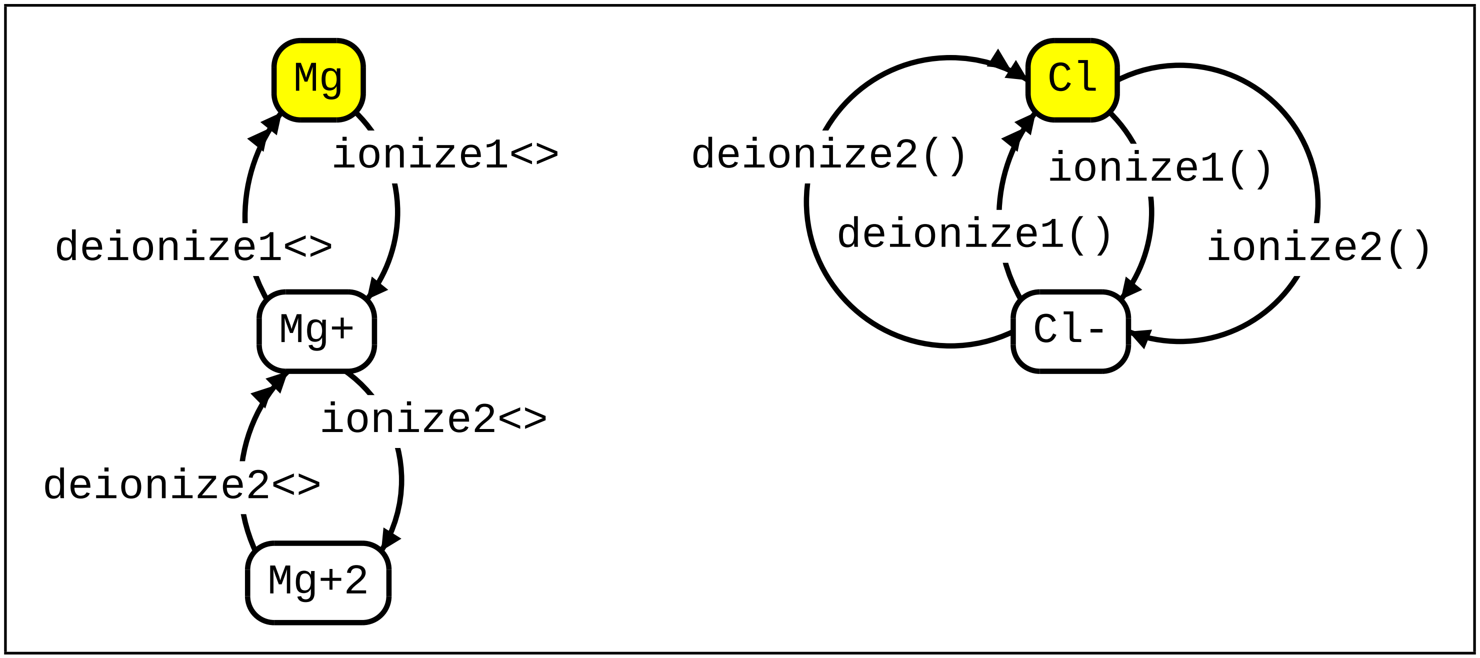
➢ Replicated output $!x\langle n\rangle$ can send a message to replicated input $!x(m)$.

Graphical Semantics



➢ Clones of P and P' are spawned, and n is assigned to m in the clone of P'.

Ionization: $Mg + 2Cl \rightleftharpoons Mg^{+2} + 2Cl^{-}$



- Choice of alternative reactions