Geometry of interaction and non-determinism

Marco Pedicini (Roma Tre University)
joint work with Mario Piazza, SNS Pisa

XXVIII Incontro di Logica AILA

Udine, 3-6 settembre 2024

Outline

- actions of streams

Outline

- actions of streams
- lambda calculus and choice as syntactic sugar

Outline

- actions of streams
- lambda calculus and choice as syntactic sugar
- Lamping and the PELCR-VM

Outline

- actions of streams
- lambda calculus and choice as syntactic sugar
- Lamping and the PELCR-VM
- molecular dynamics vs local and asynchronous computations

Outline

- actions of streams
- lambda calculus and choice as syntactic sugar
- Lamping and the PELCR-VM
- molecular dynamics vs local and asynchronous computations
- execution formula and its dynamics, non-determinism and solution *à la Gillespie*

Outline

- actions of streams
- lambda calculus and choice as syntactic sugar
- Lamping and the PELCR-VM
- molecular dynamics vs local and asynchronous computations
- execution formula and its dynamics, non-determinism and solution *à la Gillespie*

STREAMS & MACHINES

Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.

Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

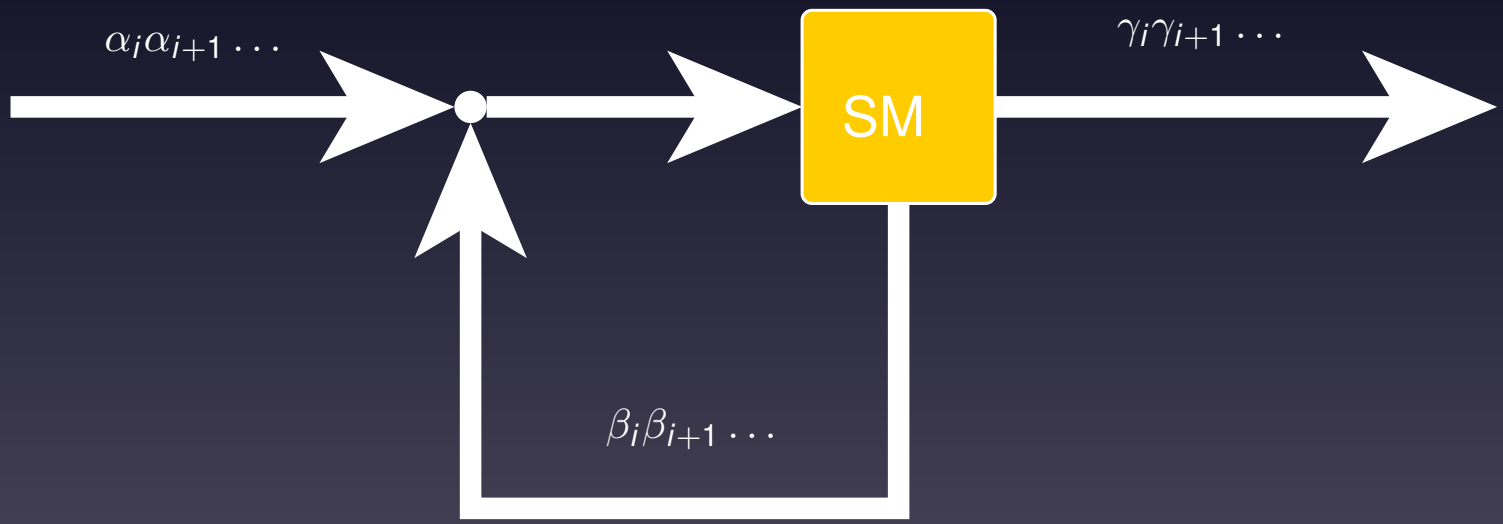
Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.

Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.

When an action is processed possibly many new actions are generated, feeding the stream.



Streams and Actions

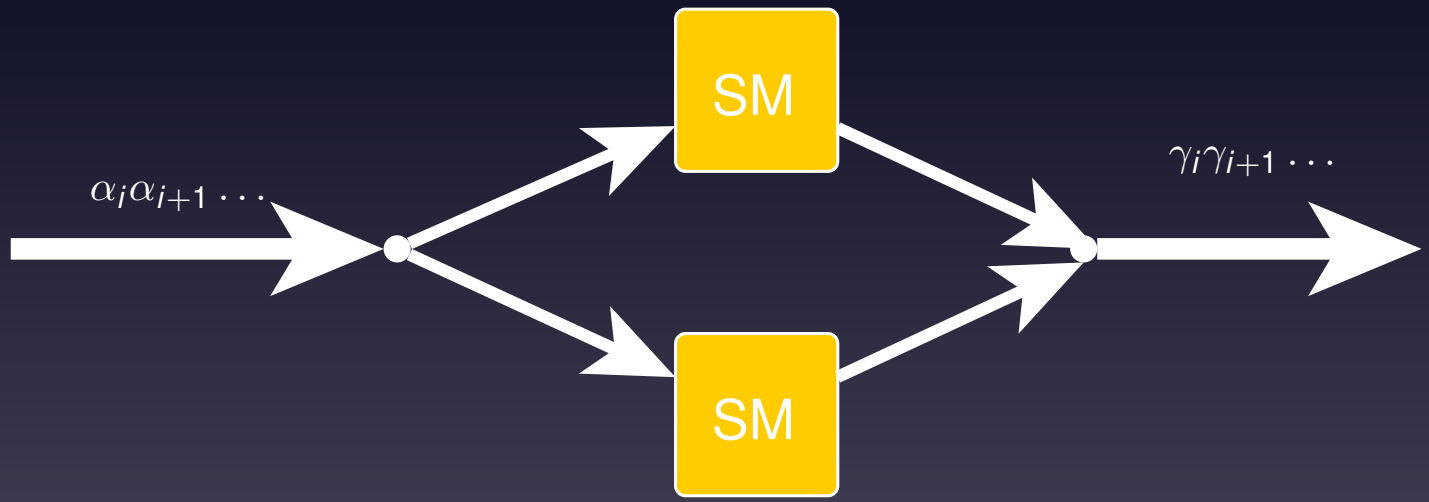
A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.

Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

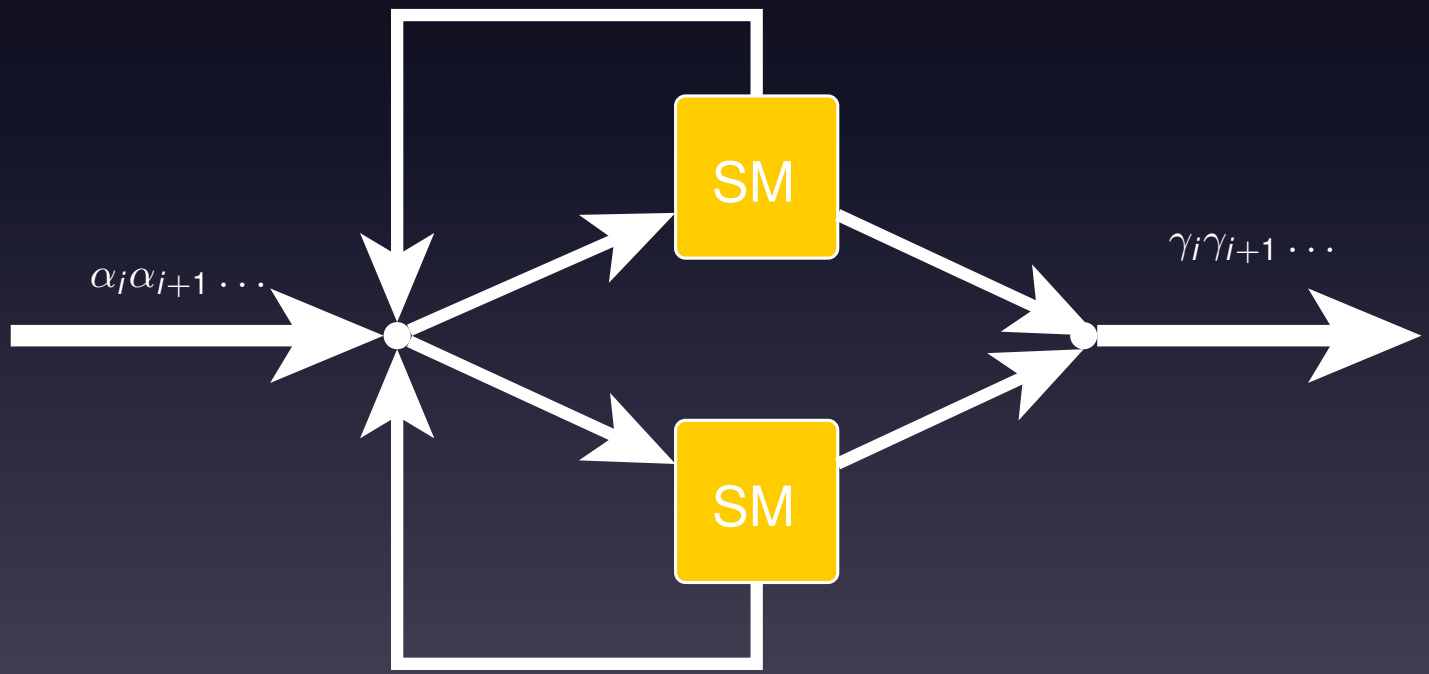
Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.



Streams and Actions

A **stream** $(\alpha_i)_i$ is a sequence of **actions** α_i of a given set A.

Stream processing machines, which inspire real world parallel computing devices like GPU's, take a stream and compute by processing action by action.



PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

S : (stack) current action(s)

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

S : (stack) current action(s)
E : (environment) part of the configuration required to execute an action

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

S : (stack) current action(s)
E : (environment) part of the configuration required to execute an action
C : (core) stream of actions to be processed

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

S : (stack) current action(s)
E : (environment) part of the configuration required to execute an action
C : (core) stream of actions to be processed
D : (dump) configuration containing all environments

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

S : (stack) current action(s)
E : (environment) part of the configuration required to execute an action
C : (core) stream of actions to be processed
D : (dump) configuration containing all environments

(S,E,C,D)

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

$$\text{POP}\ \frac{(\langle\rangle, \text{NULL}, \alpha :: C, D)}{(\alpha, \text{NULL}, C, D)}$$

get the next action from the stream

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

$$\text{POP}\ \frac{(\langle\rangle, \text{NULL}, \alpha :: C, D)}{(\alpha, \text{NULL}, C, D)}$$

get the next action from the stream

$$\text{ENV}\ \frac{(\langle\alpha\rangle, \text{NULL}, C, D) \qquad \alpha \neq \mathbf{0}}{(\langle\alpha\rangle, \text{target}(\alpha), C, D \cup \{\text{target}(\alpha)\})}$$

prepare the environment

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

$$\text{POP}\ \frac{(\langle\rangle, \text{NULL}, \alpha :: C, D)}{(\alpha, \text{NULL}, C, D)}$$

get the next action from the stream

$$\text{ENV}\ \frac{(\langle\alpha\rangle, \text{NULL}, C, D) \qquad \alpha \neq \mathbf{0}}{(\langle\alpha\rangle, \text{target}(\alpha), C, D \cup \{\text{target}(\alpha)\})}$$

prepare the environment

$$\text{HC}\ \frac{(\langle\alpha\rangle, \text{target}(\alpha), C, D)}{(\langle\rangle, \text{NULL}, C \times \text{execute}(\alpha), D \cup \{\alpha\})}$$

perform evaluation and merge in the stream new actions

PELCR-VM

The **PELCR-VM** defined in Lai et al., 2019, follows implementations in Pedicini, 1998; Pedicini e Quaglia, 2000, 2007, it is a **stream based abstract machine** where actions comes from the **Geometry of Interaction** interpretation of proof-nets (MELL fragment) and evaluation mimics optimal reduction of λ-calculus Asperti e Chroboczek, 1997; Gonthier et al., 1992; Lévy, 1978, 1980.

$$\text{POP}\ \frac{(\langle\rangle, \text{NULL}, \alpha :: C, D)}{(\alpha, \text{NULL}, C, D)}$$

get the next action from the stream

$$\text{ENV}\ \frac{(\langle\alpha\rangle, \text{NULL}, C, D) \qquad \alpha \neq \mathbf{0}}{(\langle\alpha\rangle, \text{target}(\alpha), C, D \cup \{\text{target}(\alpha)\})}$$

prepare the environment

$$\text{HC}\ \frac{(\langle\alpha\rangle, \text{target}(\alpha), C, D)}{(\langle\rangle, \text{NULL}, C \times \text{execute}(\alpha), D \cup \{\alpha\})}$$

perform evaluation and merge in the stream new actions

$$\text{NENV}\ \frac{(\langle\alpha\rangle, \text{NULL}, C, D) \qquad \alpha = \mathbf{0}}{(\langle\rangle, \text{NULL}, C, D)}$$

idle cycle (if α is the zero action)

Back and forth

Actions are paths with a source node, a target node and the weight taken in the geometry of interaction **polarised Girard's Dynamic Algebra**.

Back and forth

Actions are paths with a source node, a target node and the weight taken in the geometry of interaction **polarised Girard's Dynamic Algebra**.

Execution of an action α to be performed within a context

$$\{\beta_1, \ldots, \beta_n\}$$

generates new actions in pairs:

$$\alpha_i := \langle(\varepsilon_i, \varepsilon), (v_i, v_i'), a_i'\rangle \quad \text{and} \quad \beta_i' := \langle(\varepsilon_s, -\varepsilon), (v_s, v_i'), b_i'\rangle,$$

where ε is arbitrarily chosen in $\{+, -\}$ and v_i' is a newly generated node and if $b_i^* a \neq 0$, $a_i' b_i'^*$ is the stable form of $b_i^* a$.

Back and forth

Actions are paths with a source node, a target node and the weight taken in the geometry of interaction **polarised Girard's Dynamic Algebra**.

Execution of an action α to be performed within a context

$$\{\beta_1, \ldots, \beta_n\}$$

generates new actions in pairs:

$$\alpha_i := \langle(\varepsilon_i, \varepsilon), (v_i, v_i'), a_i'\rangle \quad \text{and} \quad \beta_i' := \langle(\varepsilon_s, -\varepsilon), (v_s, v_i'), b_i'\rangle,$$

where ε is arbitrarily chosen in $\{+, -\}$ and v_i' is a newly generated node and if $b_i^* a \neq 0$, $a_i' b_i'^*$ is the stable form of $b_i^* a$.

Essentially an action is a **polarised edge in a weighted graph**:

$$\langle(\varepsilon_t, \varepsilon_s), (v_t, v_s), w\rangle,$$

and tracks the logical structure of a path in the proof-net.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x$$

if and only if $t, u \in \Lambda_{\oplus}$ and $x \in V$ a numerable set of *variables*.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x \qquad (t)u$$

if and only if $t, u \in \Lambda_{\oplus}$ and $x \in V$ a numerable set of *variables*.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x \qquad (t)u \qquad \lambda x t$$

if and only if $t, u \in \Lambda_{\oplus}$ and $x \in V$ a numerable set of *variables*.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x \qquad (t)u \qquad \lambda x t \qquad [t \oplus u]$$

if and only if $t, u \in \Lambda_{\oplus}$ and $x \in V$ a numerable set of *variables*.

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x \qquad (t)u \qquad \lambda x t \qquad [t \oplus u]$$

Parallel Evaluation

This setting was studied in Pedicini, 1998, and linked to **parallel evaluation** of optimal lambda calculus reduction.

Extensions with syntactical sugar (integers, or generic FFI) in parallel evaluation setting where also introduced: Cosentino et al., 2006; Mackie, 1995; Pedicini e Quaglia, 2002; Pinto, 2001.

In Pedicini e Piazza, 2024, we consider optimal reduction in the setting of **non-deterministic λ-calculus**:

$$x \qquad (t)u \qquad \lambda x t \qquad [t \oplus u]$$

if and only if $t, u \in \Lambda_{\oplus}$ and $x \in V$ a numerable set of *variables*.

Sharing Graphs

A new **sound and lazy** operational interpretation of non-determinism is defined (**choice-by-need**) and we show to be compatible with shared reduction:

$$(\lambda x t)u \rightarrow_\beta t[u/x]$$

Sharing Graphs

A new **sound and lazy** operational interpretation of non-determinism is defined (**choice-by-need**) and we show to be compatible with shared reduction:

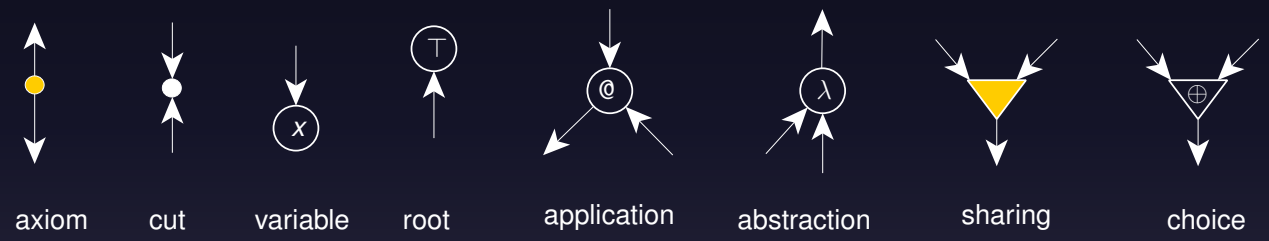
$$(\lambda x t)u \to_\beta t[u/x] \qquad ([t \oplus u])v \to_\oplus [(t)v \oplus (u)v]$$

Sharing Graphs

A new **sound and lazy** operational interpretation of non-determinism is defined (**choice-by-need**) and we show to be compatible with shared reduction:

$$(\lambda x t)u \rightarrow_\beta t[u/x] \qquad ([t \oplus u])v \rightarrow_\oplus [(t)v \oplus (u)v]$$

Shared graphs:

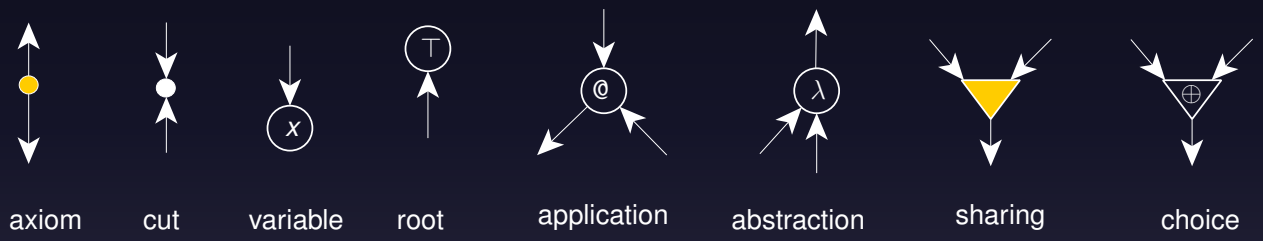


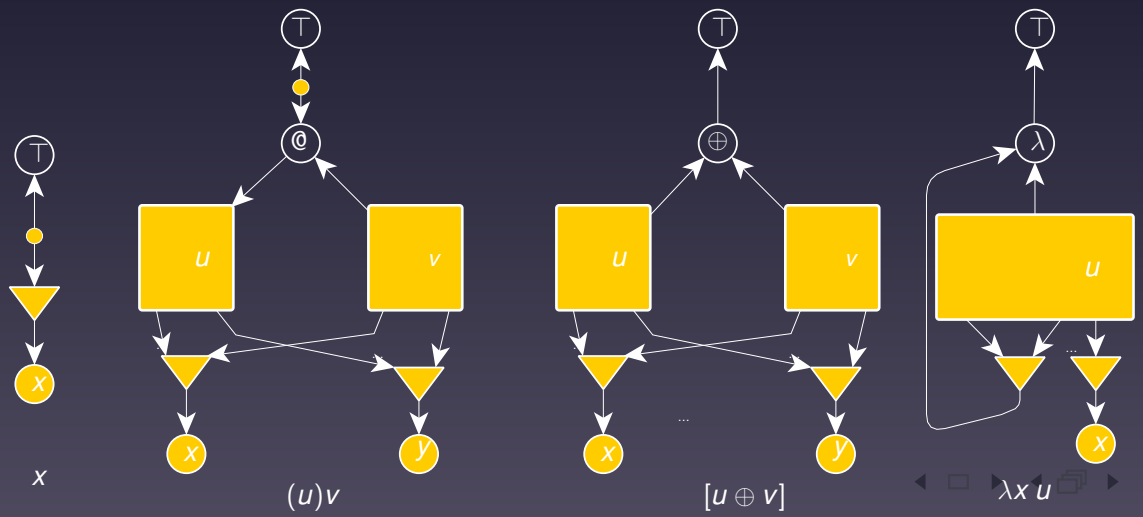
Sharing Graphs

A new **sound and lazy** operational interpretation of non-determinism is defined (**choice-by-need**) and we show to be compatible with shared reduction:

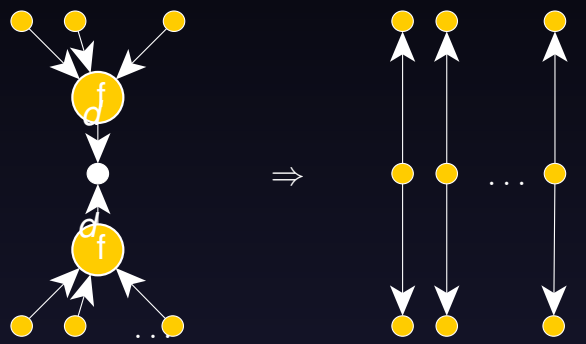
$$(\lambda x t)u \rightarrow_\beta t[u/x] \qquad ([t \oplus u])v \rightarrow_\oplus [(t)v \oplus (u)v]$$

Shared graphs:

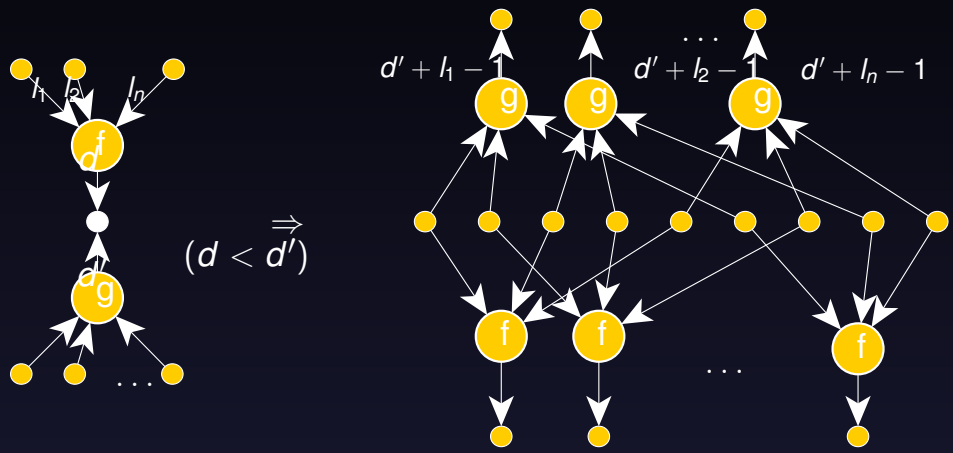




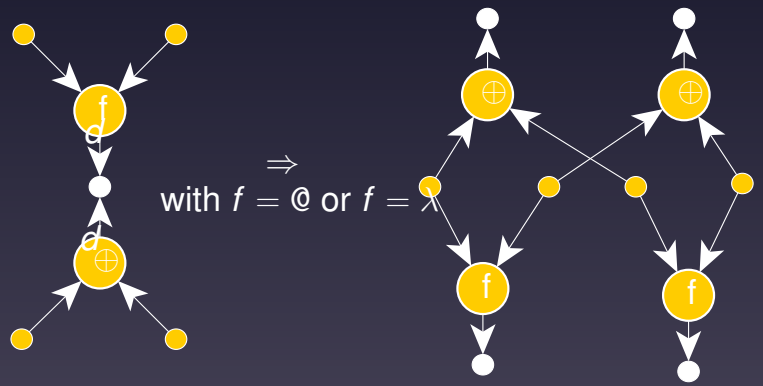
Reduction Rules



(a) annihilation

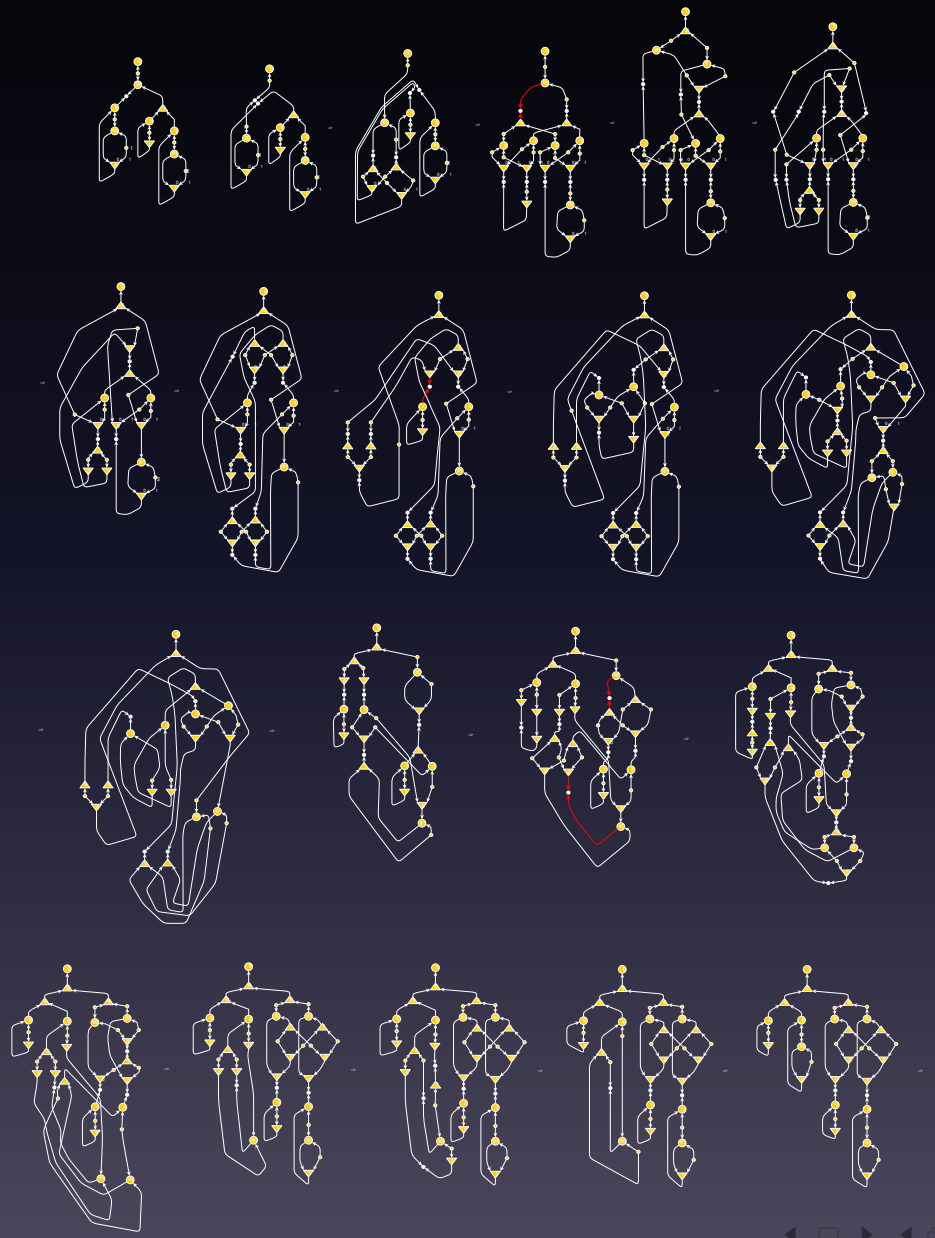


(b) commutation



(c) commutation of choice

Example of Reduction

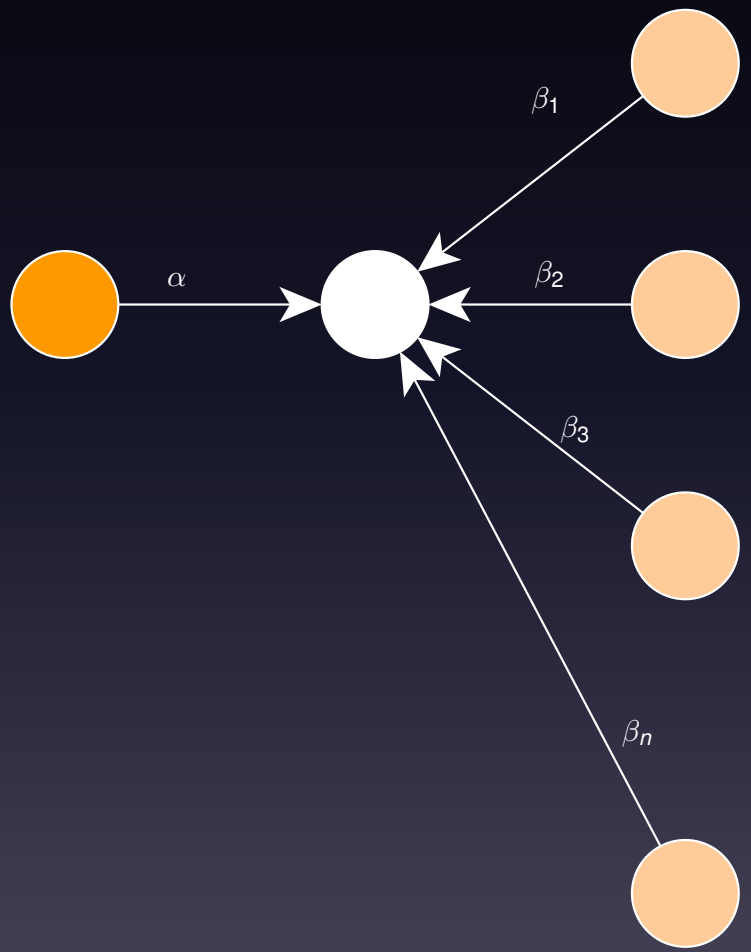


Local and Asynchronous

One only type of **active configuration**.

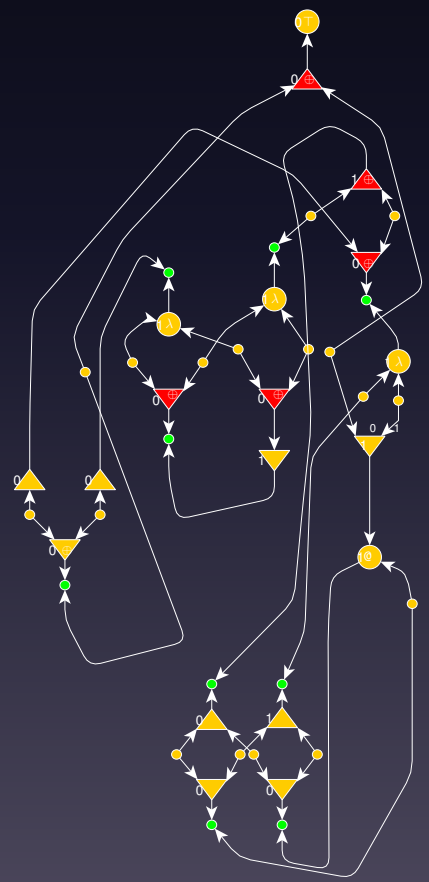
Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result.



Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result. Actions can be viewed as interacting particles: there is a collision at the **target node**, with the **context**.



Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result. Actions can be viewed as interacting particles: there is a collision at the **target node**, with the **context**. In parallel evaluation this permits to naturally **distribute active configurations** on available units.

Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result. Actions can be viewed as interacting particles: there is a collision at the **target node**, with the **context**. In parallel evaluation this permits to naturally **distribute active configurations** on available units.

Choice operator is a syntactic element establishing two separate possible **evolutions in the execution**.

Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result. Actions can be viewed as interacting particles: there is a collision at the **target node**, with the **context**. In parallel evaluation this permits to naturally **distribute active configurations** on available units.

Choice operator is a syntactic element establishing two separate possible **evolutions in the execution**.As a consequence of nesting of the choice operator, possible evolutions should be **differentially prioritized** while executed, giving rise to a **quantitative operational semantics**.

Local and Asynchronous

One only type of **active configuration**. Active configurations can be **executed concurrently** without affecting the result. Actions can be viewed as interacting particles: there is a collision at the **target node**, with the **context**. In parallel evaluation this permits to naturally **distribute active configurations** on available units.

Choice operator is a syntactic element establishing two separate possible **evolutions in the execution**.As a consequence of nesting of the choice operator, possible evolutions should be **differentially prioritized** while executed, giving rise to a **quantitative operational semantics**.

MOLECULES

Stochastic Semantics

Example of quantitative semantics for Petri Nets, Stochastic Pi-Calculus and Gene expression evolution.

Stochastic Semantics

Example of quantitative semantics for Petri Nets, Stochastic Pi-Calculus and Gene expression evolution.

Reaction Rules with reaction constants: rules with occurrence probabilities.

Stochastic Semantics

Example of quantitative semantics for Petri Nets, Stochastic Pi-Calculus and Gene expression evolution.

Reaction Rules with reaction constants: rules with occurrence probabilities.

Simulation which correctly takes in account probabilities.

Stochastic Semantics

Example of quantitative semantics for Petri Nets, Stochastic Pi-Calculus and Gene expression evolution.

Reaction Rules with reaction constants: rules with occurrence probabilities.

Simulation which correctly takes in account probabilities.

How to compute them ?

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.
4. **Generate Random Numbers:** Generate two random numbers $r_1, r_2 \in [0, 1]$.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.
4. **Generate Random Numbers:** Generate two random numbers $r_1, r_2 \in [0, 1]$.
5. **Time to Next Reaction:** Calculate the time to the next reaction $\tau = \frac{1}{a_0} \ln \frac{1}{r_1}$.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.
4. **Generate Random Numbers:** Generate two random numbers $r_1, r_2 \in [0, 1]$.
5. **Time to Next Reaction:** Calculate the time to the next reaction $\tau = \frac{1}{a_0} \ln \frac{1}{r_1}$.
6. **Determine Reaction Rule:** Find reaction j such that $\sum_{k=1}^{j-1} a_k < r_2 a_0 \leq \sum_{k=1}^{j} a_k$.

Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.
4. **Generate Random Numbers:** Generate two random numbers $r_1, r_2 \in [0, 1]$.
5. **Time to Next Reaction:** Calculate the time to the next reaction $\tau = \frac{1}{a_0} \ln \frac{1}{r_1}$.
6. **Determine Reaction Rule:** Find reaction j such that $\sum_{k=1}^{j-1} a_k < r_2 a_0 \leq \sum_{k=1}^{j} a_k$.
7. **Update:** Update the system state by executing reaction j. Update time $t = t + \tau$.

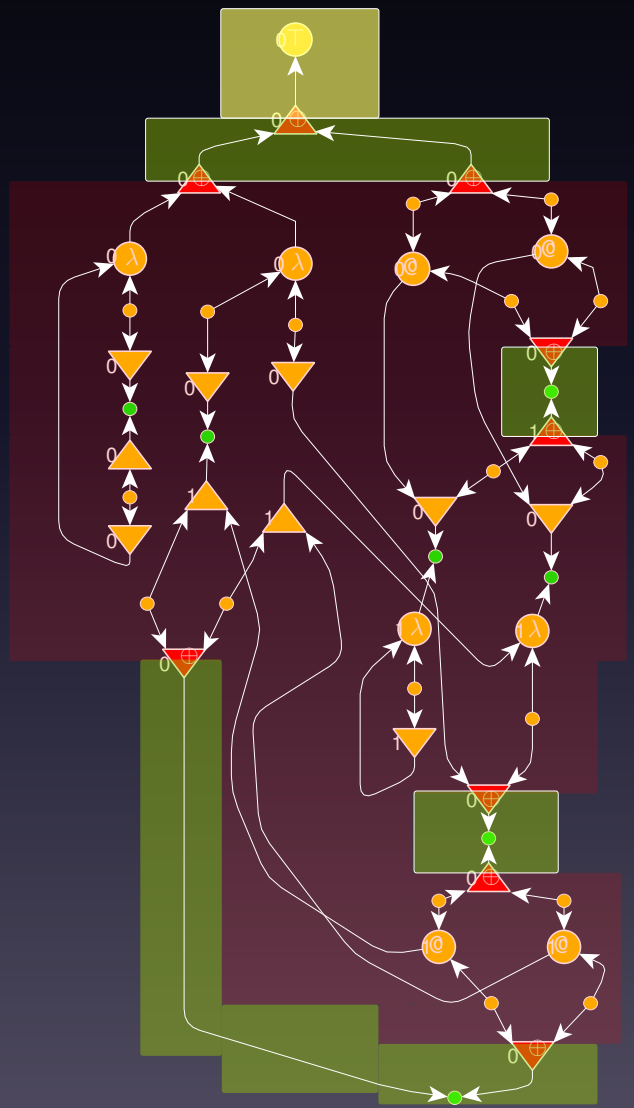
Gillespie's Algorithm

The Gillespie algorithm is a stochastic simulation method used to simulate the time evolution of a set of coupled chemical reactions within a reacting system.

1. **Initialization:** Set the initial number of molecules for each species, time $t = 0$.
2. **Calculate Propensities:** For each reaction j, calculate the propensity function $a_j = c_j \cdot h_j(X)$ where c_j is the rate constant and $h_j(X)$ is the number of possible combinations of reactant molecules.
3. **Total Propensity:** Compute the total propensity $a_0 = \sum_j a_j$.
4. **Generate Random Numbers:** Generate two random numbers $r_1, r_2 \in [0, 1]$.
5. **Time to Next Reaction:** Calculate the time to the next reaction $\tau = \frac{1}{a_0} \ln \frac{1}{r_1}$.
6. **Determine Reaction Rule:** Find reaction j such that $\sum_{k=1}^{j-1} a_k < r_2 a_0 \leq \sum_{k=1}^{j} a_k$.
7. **Update:** Update the system state by executing reaction j. Update time $t = t + \tau$.
8. **Repeat:** Repeat steps 2-7 until the desired simulation time is reached.

Probabilistic Geometry of Interaction

Open Question any action has a choice nesting depth (in the sharing graph).



- we give an interpretation of the choice nesting depth d as the rate $c_d = 2^{-d}$,
- we count the total number of active configurations at any depth d: $h(d)$
- then we apply Gillespie algorithm to schedule the current action $a_d = c_d h(d)$ and normalise against the total propensity $a = \sum_i a_i$.

Concluding remarks

- We introduced **sharing graphs** as a novel representation for the **non-deterministic lambda calculus**.
- The sharing graphs were interpreted within the framework of the **geometry of interactions** (GoI), providing a new perspective on the computational processes involved.
- We demonstrated that this interpretation can be effectively **evaluated using the PELCR-VM**, a virtual machine designed for efficient computation in this setting.
- This work opens new avenues for **further investigations** into the **operational nature of non-determinism** in computational systems.
- Future research could explore optimizing the evaluation strategies and understanding the implications of non-deterministic behaviors in broader computational contexts.

THANKS!

REFERENCES

- Asperti, A., & Chroboczek, J. (1997). Safe Operators: Brackets Closed Forever Optimizing Optimal lambda-Calculus Implementations.. *Applicable Algebra in Engineering, Communication and Computing*, *8*(6), 437–468. http://dx.doi.org/10.1007/s002000050083
- Cosentino, A., Pedicini, M., & Quaglia, F. (2006). Supporting Function Calls within PELCR [Proceedings of the First International Workshop on Developments in Computational Models (DCM 2005)]. *Electronic Notes in Theoretical Computer Science*, *135*(3), 107–117. https://doi.org/https://doi.org/10.1016/j.entcs.2005.09.025
- Gonthier, G., Abadi, M., & Lévy, J.-J. (1992). The Geometry of Optimal Lambda Reduction. *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 15–26. citeseer.ist.psu.edu/gonthier92geometry.html
- Lai, A. C., Pedicini, M., & Piazza, M. (2019). Abstract machines, optimal reduction, and streams. *Mathematical*

Structures in Computer Science, 1379–1410. https://doi.org/10.1017/S096012951900001X


Lévy, J.-J. (1978). *Réductions correctes et optimales dans le lambda-calcul* [Tesi di dottorato].


Lévy, J.-J. (1980). Optimal reductions in the lambda-calculus. *To HB Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 159–191.


Mackie, I. (1995). The geometry of interaction machine. *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 198–208.


Pedicini, M. (1998). *Exécution et Programmes* [Tesi di dottorato, Université Paris VII].


Pedicini, M., & Piazza, M. (2024, giugno). *Optimal Reduction For Non-Deterministic Lambda Calculus*.


Pedicini, M., & Quaglia, F. (2000). A parallel implementation for optimal lambda-calculus reduction. *Proceedings of the 2nd ACM SIGPLAN international conference on*

Principles and practice of declarative programming - PPDP '00. https://doi.org/10.1145/351268.351270


Pedicini, M., & Quaglia, F. (2002). Scheduling vs Communication in PELCR. In B. Monien & R. Feldmann (Cur.), *Euro-Par 2002 Parallel Processing* (pp. 648–655). Springer Berlin Heidelberg.


Pedicini, M., & Quaglia, F. (2007). PELCR: parallel environment for optimal lambda-calculus reduction. *ACM Transactions on Computational Logic (TOCL)*, *8*(3). https://doi.org/10.1145/1243996.1243997


Pinto, J. S. (2001). Parallel Implementation Models for the Lambda-Calculus Using the Geometry of Interaction. In *Typed Lambda Calculi and Applications (Abramsky, S., editor)* (pp. 385–399, Vol. 2044). Springer.