Extensional filters reveal algorithms

Guillaume Bonfante[1] Pierre Boudes[2] Jean-Yves Moyen[2]

[1]Université de Lorraine
École des Mines de Nancy

[2]Université Paris 13

Supported By the ANR-14-CE25-0005 ELICA

DICE april 2015

Introduction

Motivations

- Are "complicated" programming constructions really useful? We're still Turing-complete without them. High-order? Non-determinism? Co-arity? Multiple tapes? Large alphabets? ...
- How to compare ICC systems for Ptime?
- What is an algorithm?

Motivations

- Are "complicated" programming constructions really useful? We're still Turing-complete without them. High-order? Non-determinism? Co-arity? Multiple tapes? Large alphabets? ...
- How to compare ICC systems for Ptime?
- What is an algorithm?
- We want to ease clarity and brevity in code with powerful expressivity and complexity bounds.

Motivations

- Are "complicated" programming constructions really useful? We're still Turing-complete without them. High-order? Non-determinism? Co-arity? Multiple tapes? Large alphabets? ...
- How to compare ICC systems for Ptime?
- What is an algorithm?
- We want to ease clarity and brevity in code with powerful expressivity and complexity bounds.
- Language design: is there only one way to do it, or many?
- Program rewriting (local vs refactorisation, Felleisen 90)

Motivations

- Are "complicated" programming constructions really useful? We're still Turing-complete without them. High-order? Non-determinism? Co-arity? Multiple tapes? Large alphabets? ...
- How to compare ICC systems for Ptime?
- ~~What is an algorithm~~?
- We want to ease clarity and brevity in code with powerful expressivity and complexity bounds.
- Language design: is there only one way to do it, or many?
- Program rewriting (local vs refactorisation, Felleisen 90)

Expressivity and programming languages

Life without Cons (the second day)

First order functional programs are Turing-complete. Why using high-order?

Life without Cons (the second day)

First order functional programs are Turing-complete. Why using high-order?
The answer is in Jones' result about `cons` free programs:

- First order `cons` free: PTIME
- Second order `cons` free: EXPTIME
- High order `cons` free: ELEMENTARY

Life without Cons (the second day)

First order functional programs are Turing-complete. Why using high-order?
The answer is in Jones' result about `cons` free programs:

- First order `cons` free: PTIME
- Second order `cons` free: EXPTIME
- High order `cons` free: ELEMENTARY

High order is "more expressive" than first order, but Turing-completeness hide this.

Compilation

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}$ (syntactic)

$\mathtt{L}$

$\mathtt{M}$

Compilation

- Programming language: `Pgms` $\supset$ `L`, `M` (syntactic)
- Functions over a given domain: FUN

`L`

`M`

Compilation

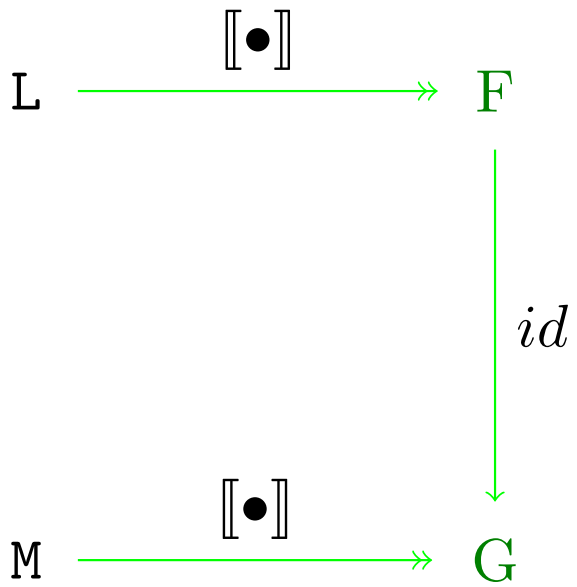
- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}$ (syntactic)
- Functions over a given domain: $\textsc{Fun}$
- Semantics: $\mathtt{Pgms} \xrightarrow{[\![\bullet]\!]} \textsc{Fun}$

$\mathtt{L}$

$\mathtt{M}$

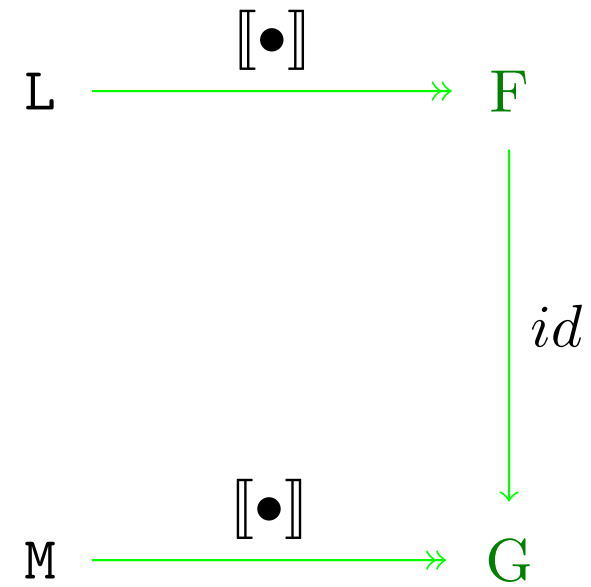
Compilation

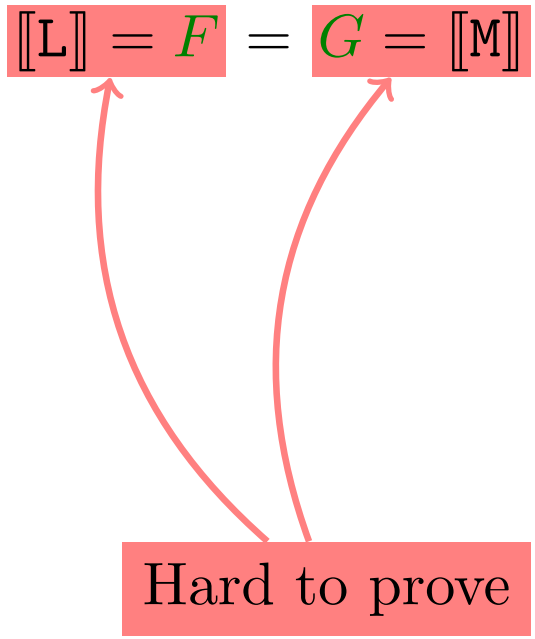
- Programming language: Pgms $\supset$ L, M (syntactic)
- Functions over a given domain: Fun
- Semantics: $\mathtt{Pgms} \xrightarrow{[\![\bullet]\!]} \textsc{Fun}$ $\quad [\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$



Compilation

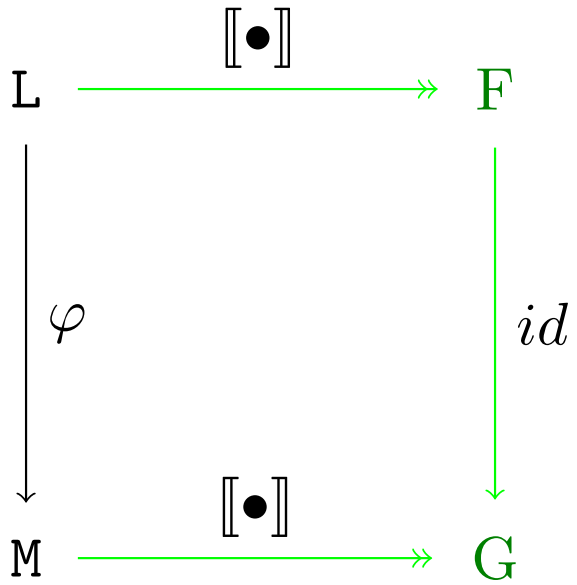
- Programming language: `Pgms` $\supset$ `L`, `M` (syntactic)
- Functions over a given domain: Fun
- Semantics: $\texttt{Pgms} \xrightarrow{[\![\bullet]\!]} \textsc{Fun}$





Compilation

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}$ (syntactic)
- Functions over a given domain: $\textsc{Fun}$
- Semantics: $\mathtt{Pgms} \xrightarrow{[\![\bullet]\!]} \textsc{Fun}$ $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- Compilation $\varphi : \mathtt{L} \to \mathtt{M}$, $[\![\varphi(\mathtt{p})]\!] = [\![\mathtt{p}]\!]$ Semantics preserving



Filtering

Hypothesis

- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$

Filtering

Hypothesis

- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- "Filter": set of programs $\mathtt{Q} \subset \mathtt{Pgms}$ (syntactic criterion)
- Filtering: $\mathtt{L'} = \mathtt{L} \bigcap \mathtt{Q}$, $\mathtt{M'} = \mathtt{M} \bigcap \mathtt{Q}$

Filtering

Hypothesis

- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- "Filter": set of programs $\mathtt{Q} \subset \mathtt{Pgms}$ (syntactic criterion)
- Filtering: $\mathtt{L'} = \mathtt{L} \bigcap \mathtt{Q}$, $\mathtt{M'} = \mathtt{M} \bigcap \mathtt{Q}$
- Breaking semantic equality: $[\![\mathtt{L'}]\!] = F' \supsetneq G' = [\![\mathtt{M'}]\!]$

Filtering

Hypothesis

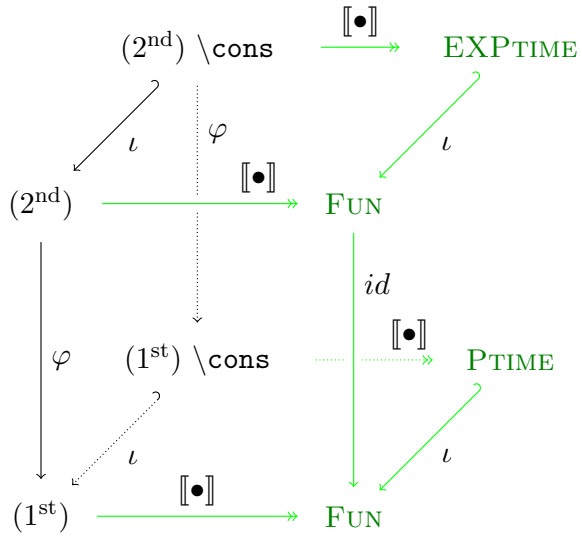
- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- "Filter": set of programs $\mathtt{Q} \subset \mathtt{Pgms}$ (syntactic criterion)
- Filtering: $\mathtt{L'} = \mathtt{L} \bigcap \mathtt{Q}$, $\mathtt{M'} = \mathtt{M} \bigcap \mathtt{Q}$
- Breaking semantic equality: $[\![\mathtt{L'}]\!] = F' \supsetneq G' = [\![\mathtt{M'}]\!]$

Conclusion

There is no filter-preserving compilation: $\mathtt{p} \in \mathtt{Q} \Leftrightarrow \varphi(\mathtt{p}) \in \mathtt{Q}$

Conversely, if there exists a filter preserving compilation, then $F' = G'$.

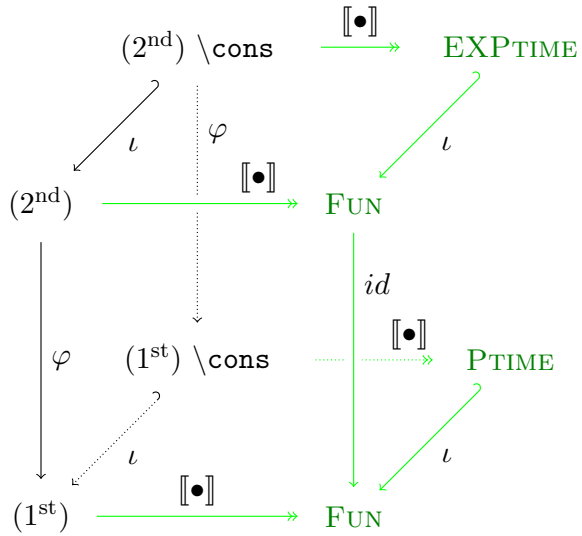
High Order



$\texttt{Pgms}$ = high order TRS

$\texttt{L} = (2^{\text{nd}})$ $\quad$ $\texttt{M} = (1^{\text{st}})$ $\texttt{Q} = \backslash\texttt{cons}$

High Order



$\texttt{Pgms}$ = high order TRS

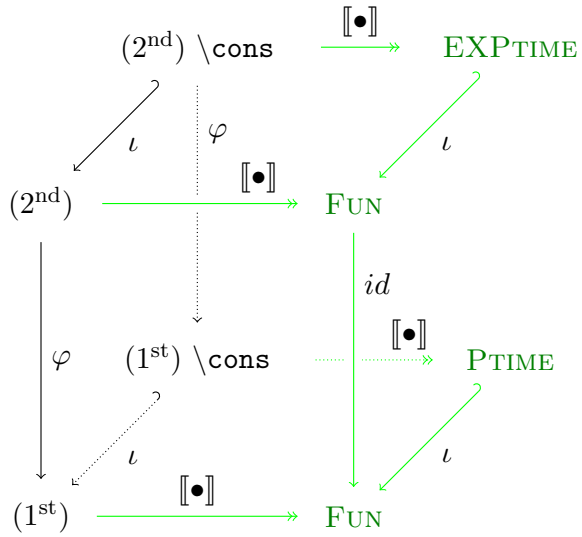
$\texttt{L} = (2^{\text{nd}}) \qquad \texttt{M} = (1^{\text{st}})\ \texttt{Q} = \backslash\texttt{cons}$

$[\![(2^{\text{nd}})]\!] = \text{FUN} = [\![(1^{\text{st}})]\!]$

$[\![(2^{\text{nd}})\ \backslash\ \texttt{cons}]\!] = \text{EXPTIME}$

$[\![(1^{\text{st}})\ \backslash\ \texttt{cons}]\!] = \text{PTIME}$

High Order



$\texttt{Pgms}$ = high order TRS

$\texttt{L} = (2^{\text{nd}}) \qquad \texttt{M} = (1^{\text{st}})\ \texttt{Q} = \texttt{\cons}$

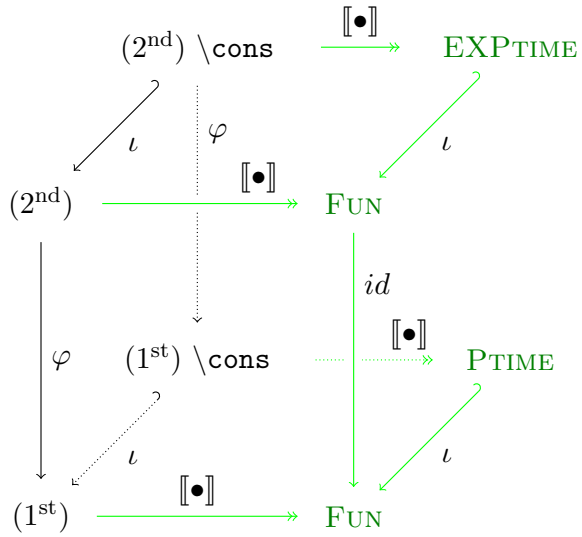
$[\![(2^{\text{nd}})]\!] = \text{FUN} = [\![(1^{\text{st}})]\!]$

$[\![(2^{\text{nd}}) \setminus \texttt{cons}]\!] = \text{EXPTIME}$

$[\![(1^{\text{st}}) \setminus \texttt{cons}]\!] = \text{PTIME}$

EXPTIME $\neq$ PTIME, hence there exists no compilation from (2^{nd}) to (1^{st}) preserving $\texttt{\cons}$.

High Order



$\texttt{Pgms}$ = high order TRS

$\texttt{L} = (2^{\text{nd}}) \qquad \texttt{M} = (1^{\text{st}})\ \texttt{Q} = \texttt{\cons}$

$[\![(2^{\text{nd}})]\!] = \text{FUN} = [\![(1^{\text{st}})]\!]$

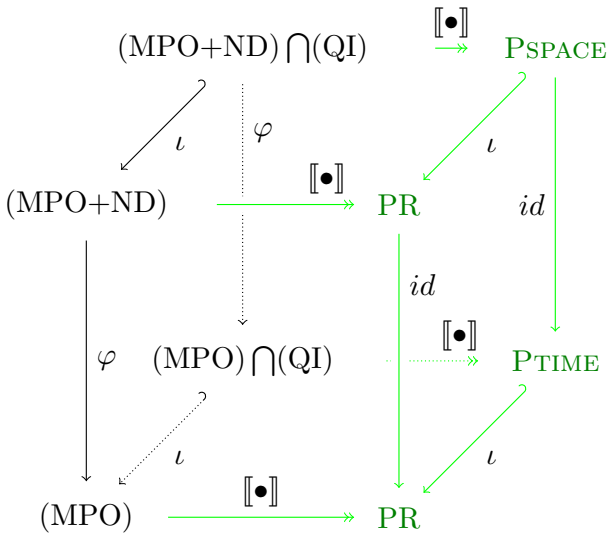
$[\![(2^{\text{nd}}) \setminus \texttt{cons}]\!] = \text{EXPTIME}$

$[\![(1^{\text{st}}) \setminus \texttt{cons}]\!] = \text{PTIME}$

EXPTIME $\neq$ PTIME, hence there exists no compilation from (2^{nd}) to (1^{st}) preserving $\texttt{\cons}$.

High order programs are more expressive than first order ones.

Non-Determinism

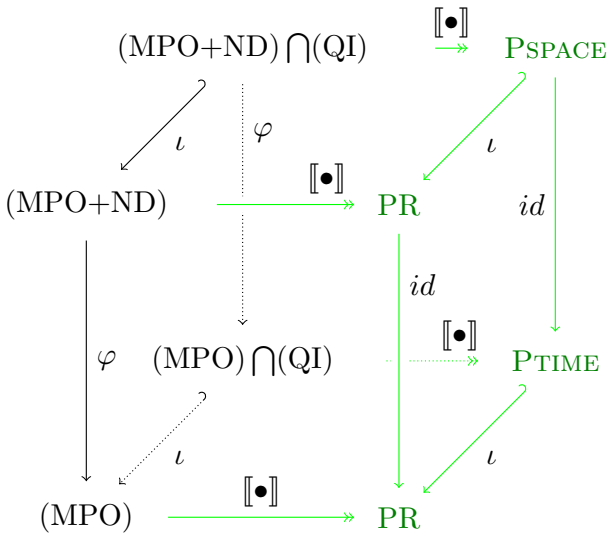


$\mathtt{Pgms}$ = ND first order TRS

$\mathtt{L} = (\mathrm{MPO{+}ND})$ $\quad$ $\mathtt{M} = (\mathrm{MPO})$

$\mathtt{Q} = (\mathrm{QI})$

Non-Determinism



$\mathtt{Pgms}$ = ND first order TRS

$\mathtt{L} = (\text{MPO+ND})$ $\quad$ $\mathtt{M} = (\text{MPO})$

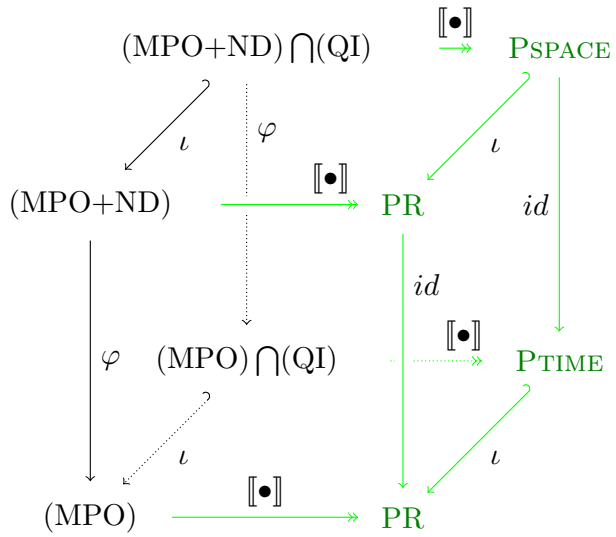
$\mathtt{Q} = (\text{QI})$

$[\![(\text{MPO+ND})]\!] = PR = [\![(\text{MPO})]\!]$

$[\![(\text{MPO+ND}) \bigcap (\text{QI})]\!] = \text{PSPACE}$

$[\![(\text{MPO}) \bigcap (\text{QI})]\!] = \text{PTIME}$

Non-Determinism



Pgms = ND first order TRS

L = (MPO+ND) M = (MPO)

Q = (QI)

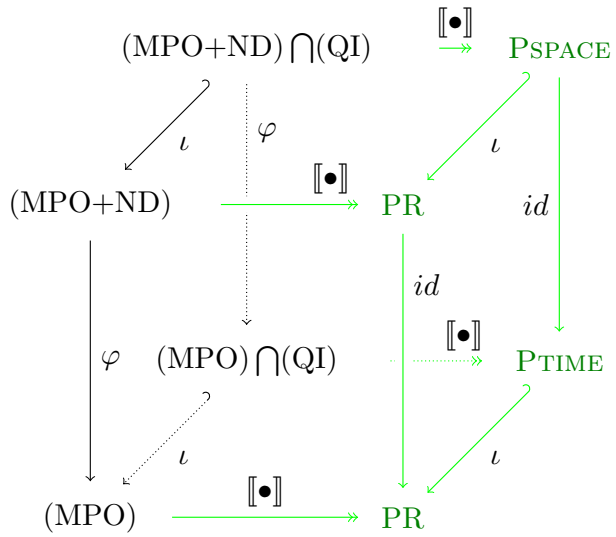
$[\![(\text{MPO+ND})]\!] = PR = [\![(\text{MPO})]\!]$

$[\![(\text{MPO+ND}) \bigcap (\text{QI})]\!] = \text{PSPACE}$

$[\![(\text{MPO}) \bigcap (\text{QI})]\!] = \text{PTIME}$

PSPACE $\neq$ PTIME (maybe), hence there exists no compilation from (MPO+ND) to (MPO) preserving (QI).

Non-Determinism



Pgms = ND first order TRS
L = (MPO+ND) M = (MPO)
Q = (QI)

$[\![(\text{MPO+ND})]\!] = PR = [\![(\text{MPO})]\!]$
$[\![(\text{MPO+ND}) \bigcap (\text{QI})]\!] = \text{PSPACE}$
$[\![(\text{MPO}) \bigcap (\text{QI})]\!] = \text{PTIME}$

PSPACE $\neq$ PTIME (maybe), hence there exists no compilation from (MPO+ND) to (MPO) preserving (QI).

Non-deterministic programs are more expressive than first order ones.

Algorithms

What is an Algorithm?

- Solid (mathematical) theory of functions.
- No good theory of algorithms
  (Gurevich's thesis: ASM $\equiv$ algorithm).

What is an Algorithm?

- Solid (mathematical) theory of functions.
- No good theory of algorithms
  (Gurevich's thesis: ASM $\equiv$ algorithm).

What's sure:

- Two programs may implement the same algorithm.
- Two algorithms may compute the same function.

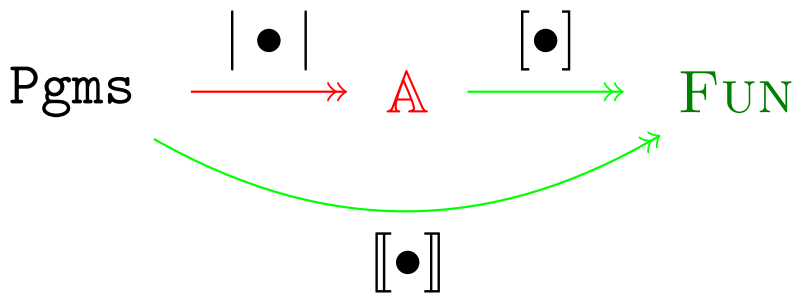
What is an Algorithm?

- Solid (mathematical) theory of functions.
- No good theory of algorithms
  (Gurevich's thesis: ASM $\equiv$ algorithm).

What's sure:

- Two programs may implement the same algorithm.
- Two algorithms may compute the same function.

Algorithms lay somewhere between programs and functions.



$$[|\mathtt{p}|] = [\![\mathtt{p}]\!]$$

The Algorithmic Level

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}, \mathtt{Q}$ (syntactic)

The Algorithmic Level

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}, \mathtt{Q}$ (syntactic)
- Algorithms (?): $\mathbb{A}$ Computable functions: $\textsc{Fun}$
- Semantics: $\mathtt{Pgms} \xrightarrow{|\bullet|} \mathbb{A} \xrightarrow{[\bullet]} \textsc{Fun}$

  Filters now respect algorithms:
  if $|\mathtt{p}_1| = |\mathtt{p}_2|$ then $\mathtt{p}_1 \in Q \Leftrightarrow \mathtt{p}_2 \in Q$

The Algorithmic Level

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}, \mathtt{Q}$ (syntactic)
- Algorithms (?): $\mathbb{A}$ Computable functions: $\textsc{Fun}$
- Semantics: $\mathtt{Pgms} \xrightarrow{|\bullet|} \mathbb{A} \xrightarrow{[\bullet]} \textsc{Fun}$

  Filters now respect algorithms:
  if $|\mathtt{p}_1| = |\mathtt{p}_2|$ then $\mathtt{p}_1 \in Q \Leftrightarrow \mathtt{p}_2 \in Q$
- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$

The Algorithmic Level

- Programming language: $\mathtt{Pgms}$ $\supset \mathtt{L}, \mathtt{M}, \mathtt{Q}$ (syntactic)
- Algorithms (?): $\mathbb{A}$ Computable functions: $\textsc{Fun}$
- Semantics: $\mathtt{Pgms} \xrightarrow{|\bullet|} \mathbb{A} \xrightarrow{[\bullet]} \textsc{Fun}$

  Filters now respect algorithms:
  if $|\mathtt{p}_1| = |\mathtt{p}_2|$ then $\mathtt{p}_1 \in Q \Leftrightarrow \mathtt{p}_2 \in Q$
- $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- Filtering: $\mathtt{L'} = \mathtt{L} \bigcap \mathtt{Q}$, $\mathtt{M'} = \mathtt{M} \bigcap \mathtt{Q}$
- Breaking semantic equality: $[\![\mathtt{L'}]\!] = F' \supsetneq G' = [\![\mathtt{M'}]\!]$

Filtering with algorithms

Hypothesis

- $|\mathtt{L}| = X = Y = |\mathtt{M}|$ so $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- Filter definition: $\mathtt{p}_1 \in Q$ and $|\mathtt{p}_1| = |\mathtt{p}_2| \Rightarrow \mathtt{p}_2 \in Q$

Filtering with algorithms

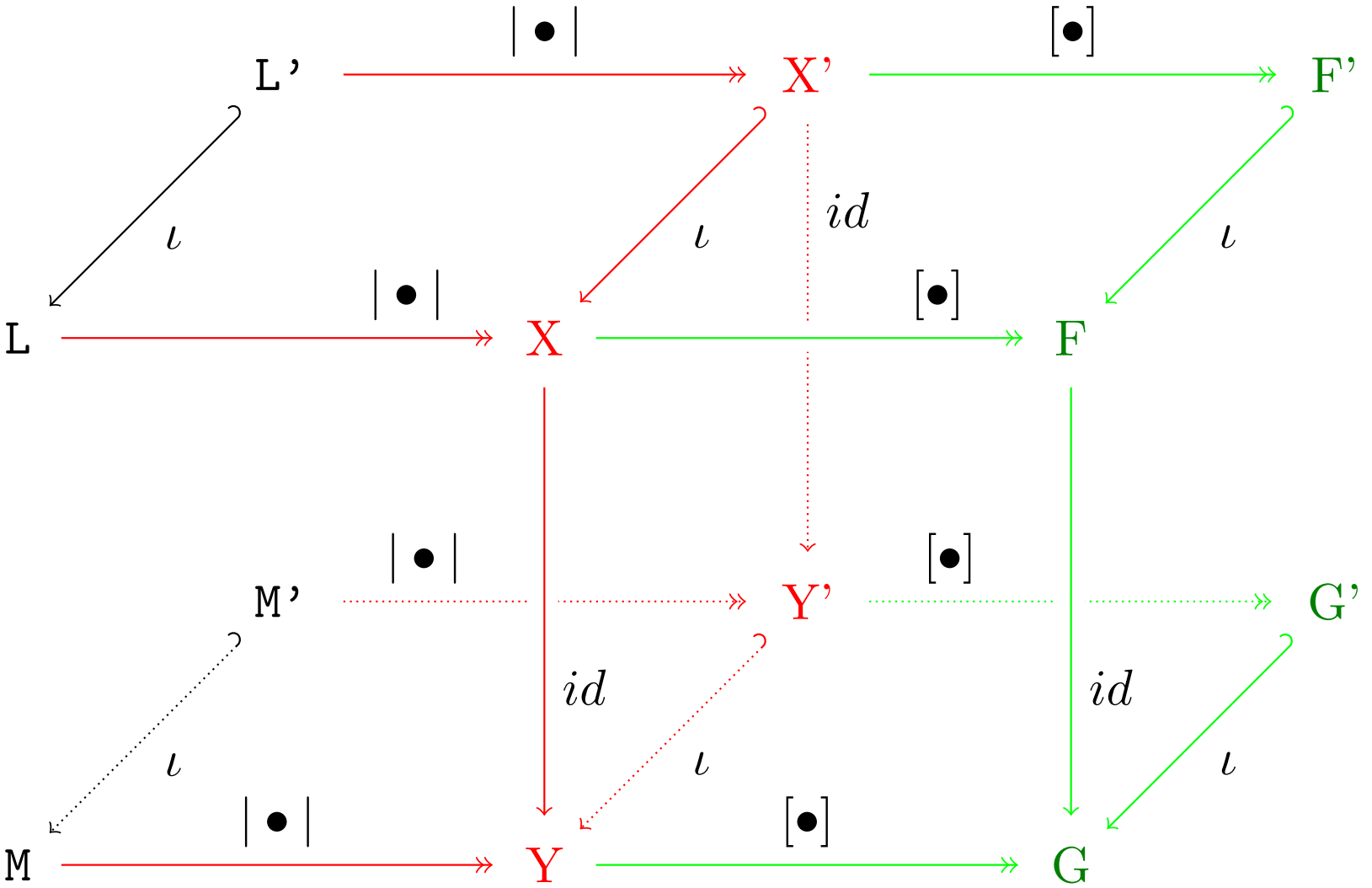
Hypothesis

- $|\mathtt{L}| = X = Y = |\mathtt{M}|$ so $[\![\mathtt{L}]\!] = F = G = [\![\mathtt{M}]\!]$
- Filter definition: $\mathtt{p}_1 \in Q$ and $|\mathtt{p}_1| = |\mathtt{p}_2| \Rightarrow \mathtt{p}_2 \in Q$

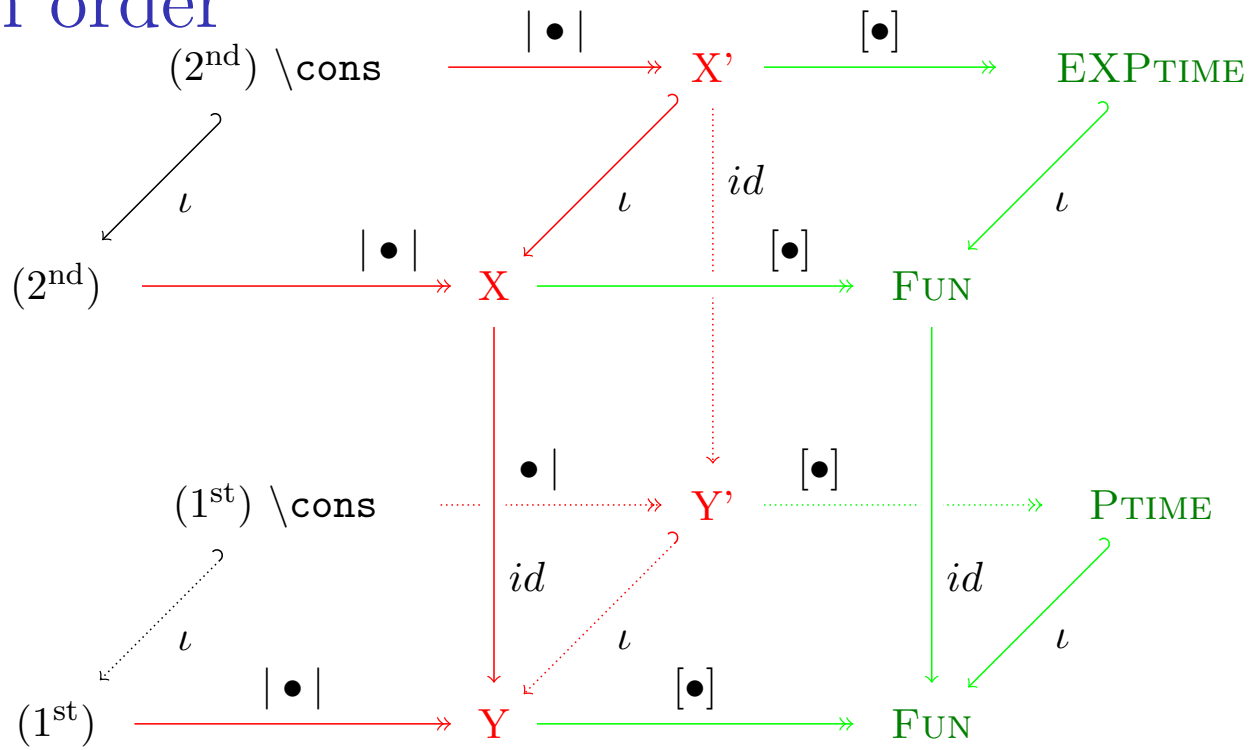
Conclusion

If there exists a filter $\mathtt{Q}$ such that $[\![\mathtt{L} \bigcap Q]\!] = F' \neq G' = [\![\mathtt{M} \bigcap Q]\!]$ then F and G are not algorithmically equivalent.

The big picture

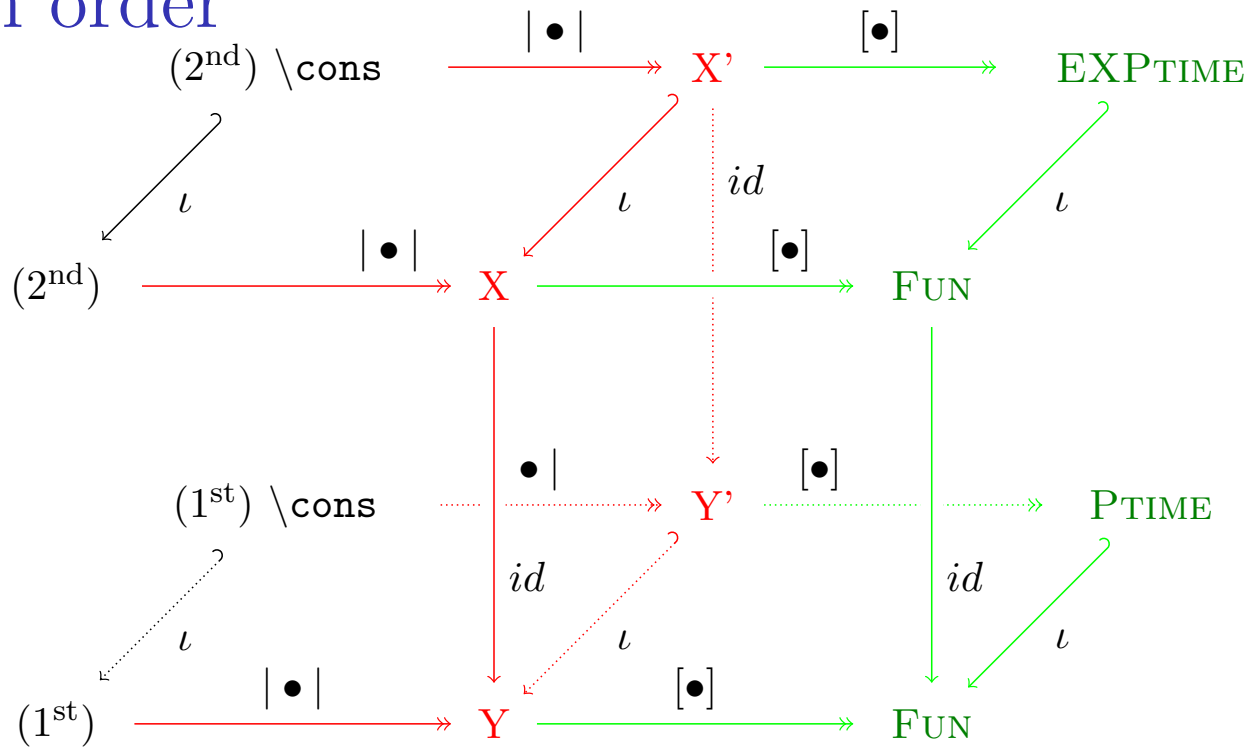


High order



Algorithms respect `cons`-free.

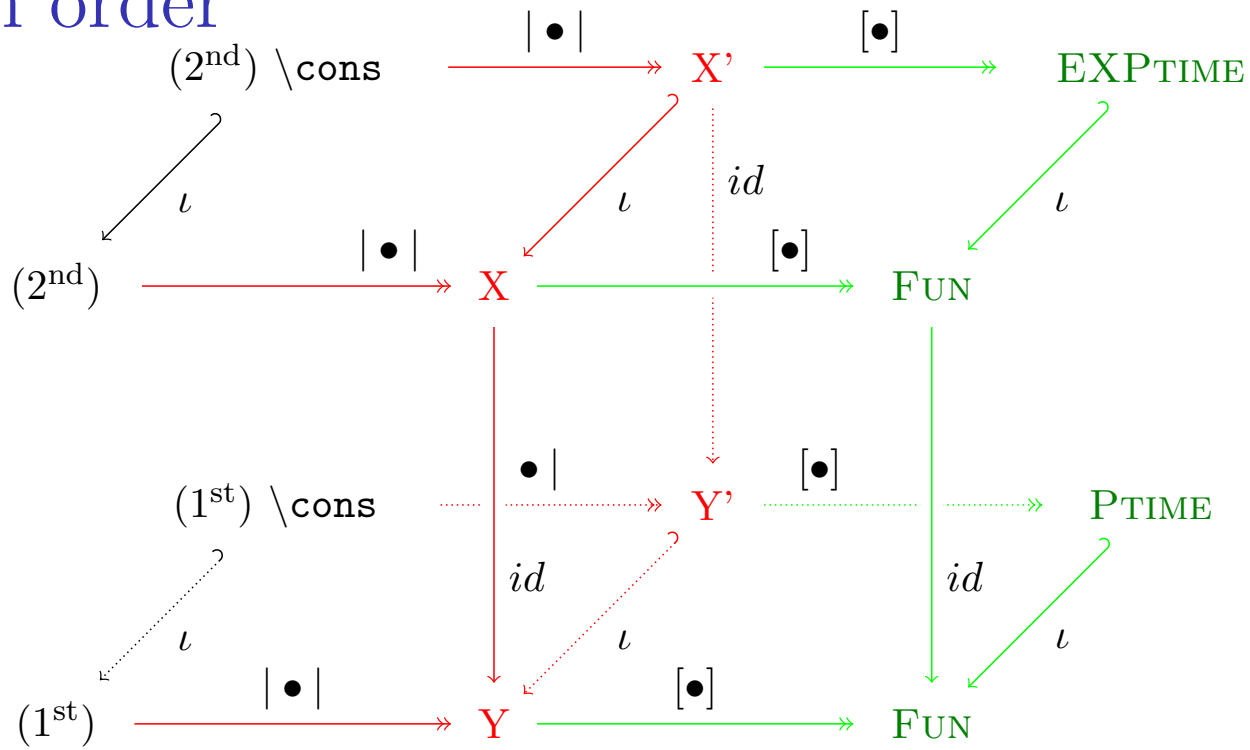
High order



Algorithms respect `cons`-free.

There are `cons`-free high-order algorithms with no first order counterpart.

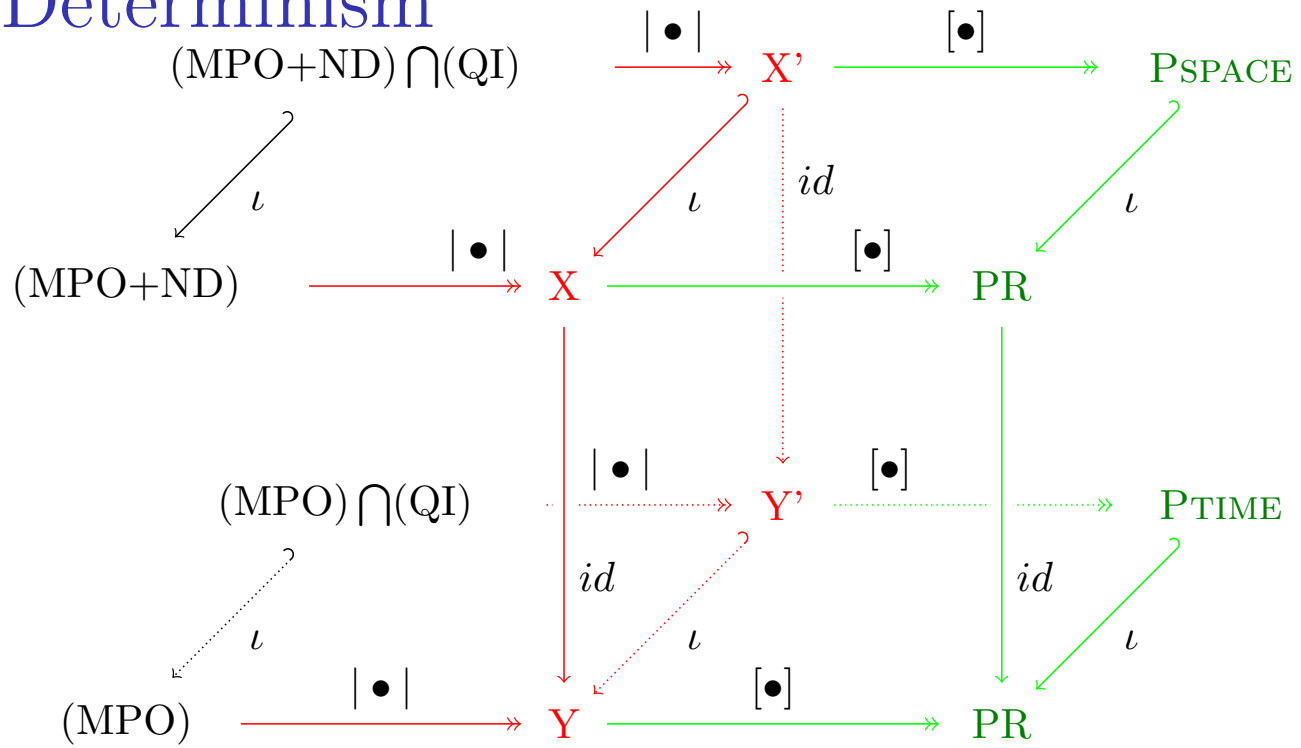
High order



Algorithms respect **cons**-free.

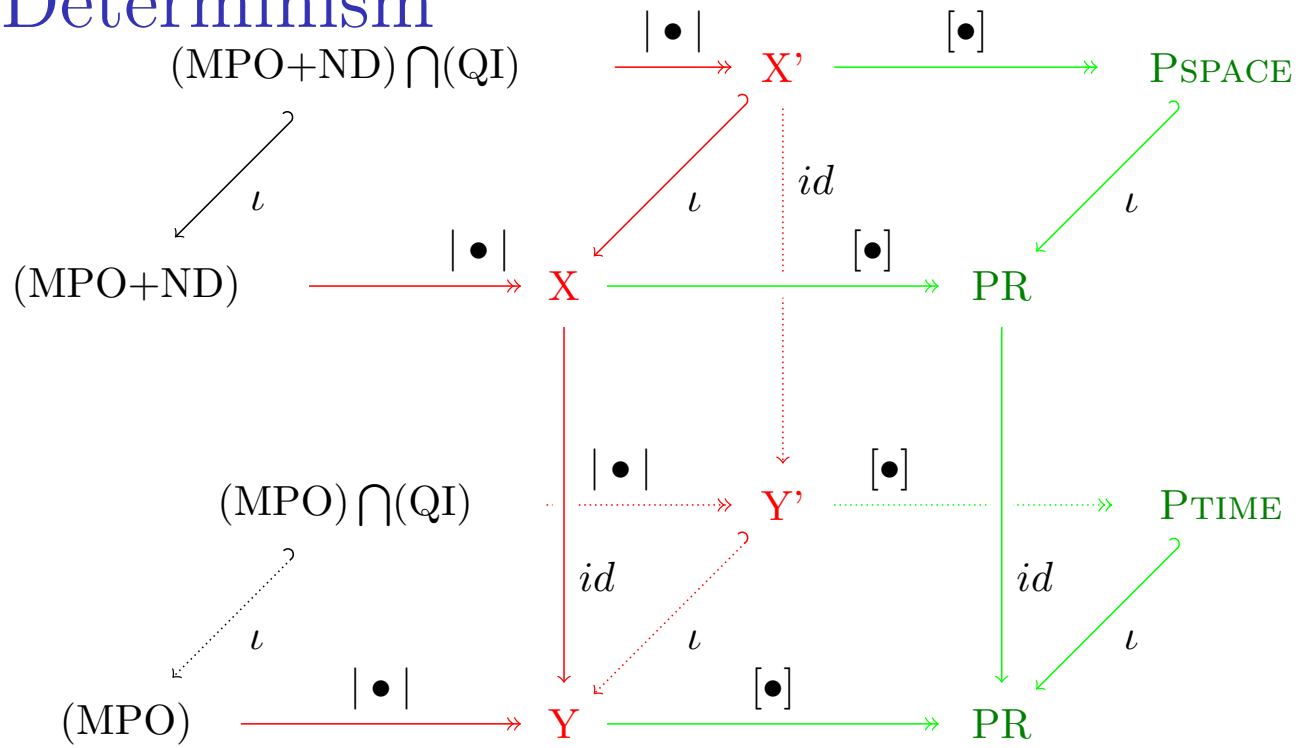
There are **cons**-free high-order algorithms with no first order counterpart.

Non Determinism



Algorithms respect QI.

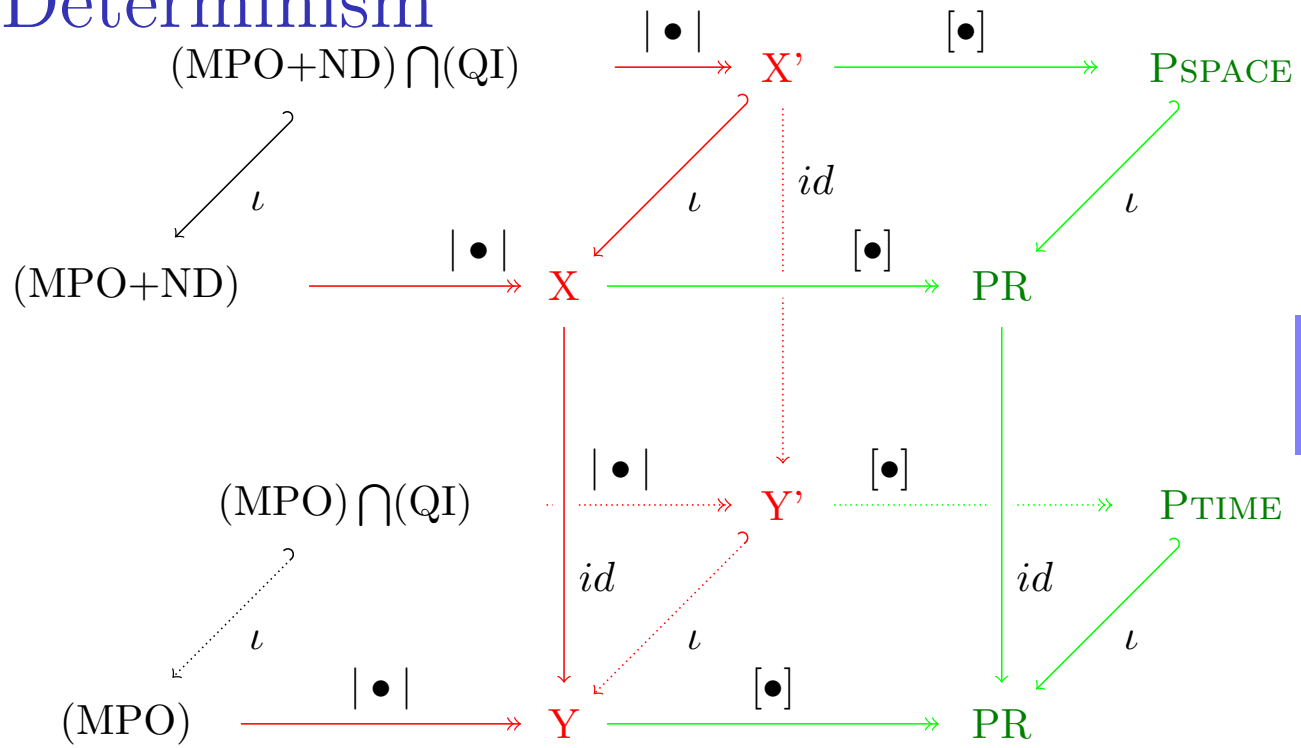
Non Determinism



Algorithms respect QI.

There are Non-deterministic algorithms admitting a Quasi-Interpretation with no deterministic counterpart.

Non Determinism



Algorithms respect QI.

There are Non-deterministic algorithms admitting a Quasi-Interpretation with no deterministic counterpart.

Chemistry

A chemical version of this:

Conclusion

Conclusion

- The framework is generic and could be applied to many cases, with little assumptions about "algorithms"
- But we still need to find a good filter and prove:
  $[\![\mathtt{L}]\!] = [\![\mathtt{M}]\!]$
  $[\![\mathtt{L'}]\!] \neq [\![\mathtt{M'}]\!]$
- Reuse 20 years of ICC to get more results