Geometry of Interaction
with Applications to ICC

Ugo Dal Lago

Logic and Interactions, February 7th 2012

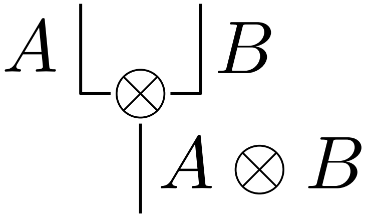
Geometry of Interaction

- A dynamic, interactive approach to interpreting the rules of (linear) logic.
- Many possible ways of presenting GoI.
  - Operator algebras [Girard1987,Girard2011];
  - Categorical constructions [JSV1997, AHS2002];
  - An algebra of weights [DR1992, DR1993];
  - Token Machines [DR1996];
  - Context semantics [GAL1992];
  - ...
- Here, we are specially interested in GoI in its concrete, algorithmic incarnations, namely **token machines** and **context semantics**.
  - Tool to prove properties of programs and proofs.
  - Model of computation.

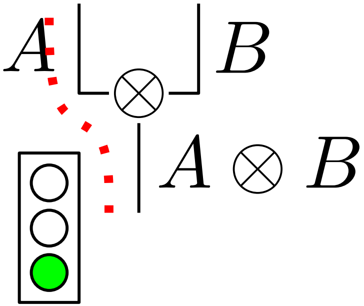
Part I

Token Machines

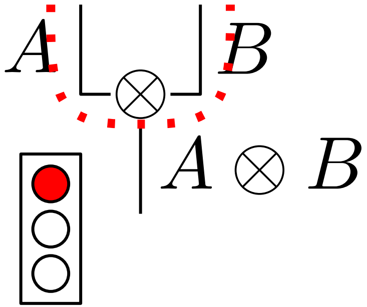
Straight Paths



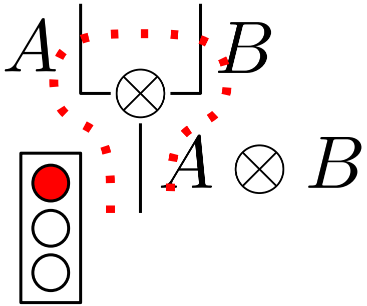
Straight Paths



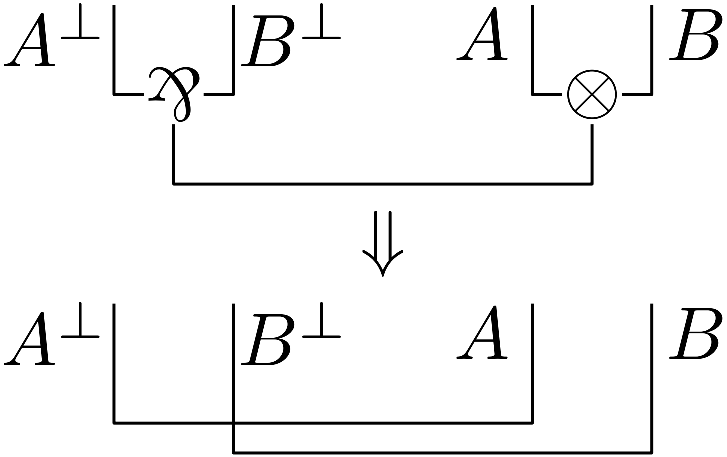
Straight Paths



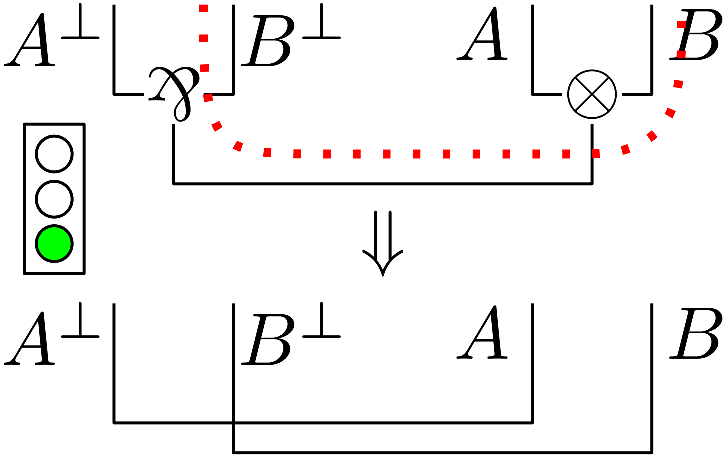
Straight Paths



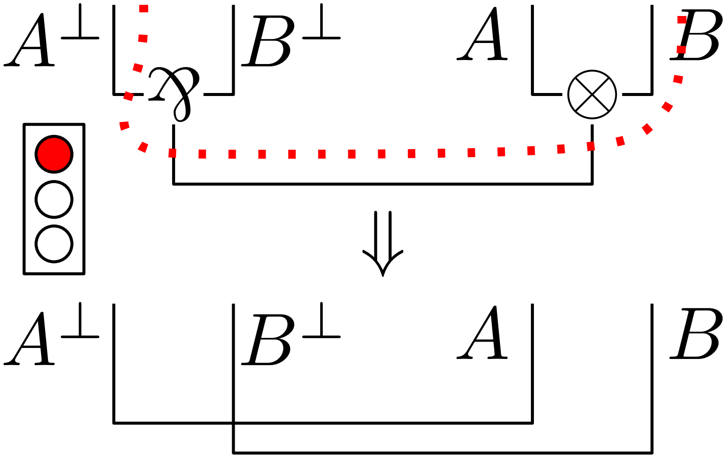
Persistent Paths



Persistent Paths

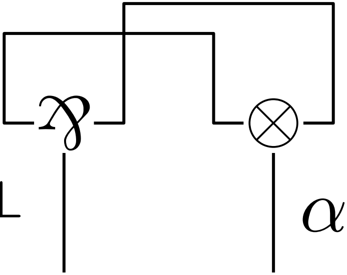


Persistent Paths



Persistency Matters

true

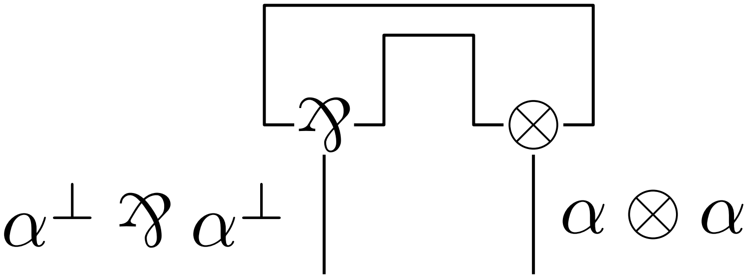


$\alpha^{\perp} \parr \alpha^{\perp}$

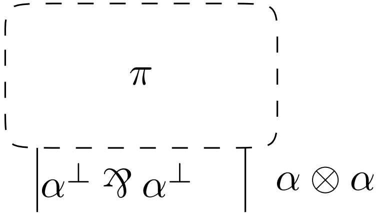
$\alpha \otimes \alpha$

Persistency Matters

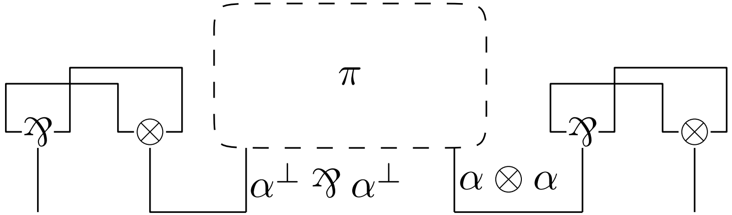
false



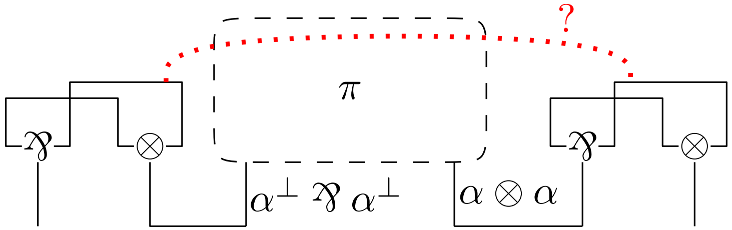
Persistency Matters



Persistency Matters



Persistency Matters

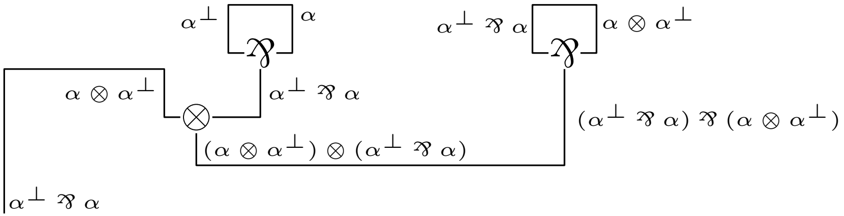


How To Capture Persistency?

- Every proof π corresponds to an automaton $\mathcal{A}_\pi$.
- **States** of $\mathcal{A}_\pi$ are elements of $\mathcal{S}_\pi = E_\pi \times \mathcal{C}$, where
  - E_π are the edges of π;
  - $\mathcal{C}$ are *contexts* (i.e., formulas with an hole) in the underlying logic **MLL**.
- The **transition function** $\to_\pi$ is a binary relation on $\mathcal{S}_\pi$ which is bideterministic.
  - Whenever $(e, C) \to_\pi (f, D)$, (e, f) forms a short direct path.
- How is $\to_\pi$ is defined?
  - We should preserve the following **invariant** along a computation: either $F(e) = C[\alpha]$ or $F(e) = C[\alpha^\perp]$. Let $Atom(e, C) = \alpha$.
  - Moreover, $(e, C) \to_\pi (f, D)$ then $Atom(e, C) = Atom(f, D)$.
- This works for proofs in propositional **MLL**.

How to Capture Persistency? - Example

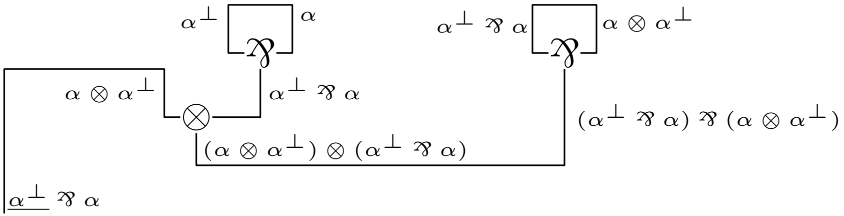
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

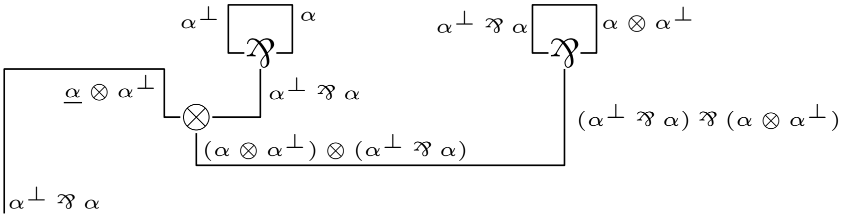
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\underline{\alpha^\perp} \parr \alpha$

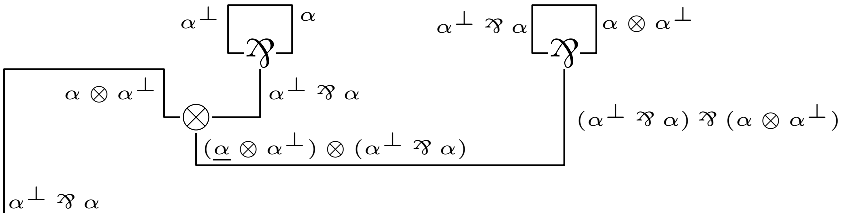
How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$


$\alpha^\perp$
α
$\parr$
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\parr$
$\underline{\alpha} \otimes \alpha^\perp$
$\otimes$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

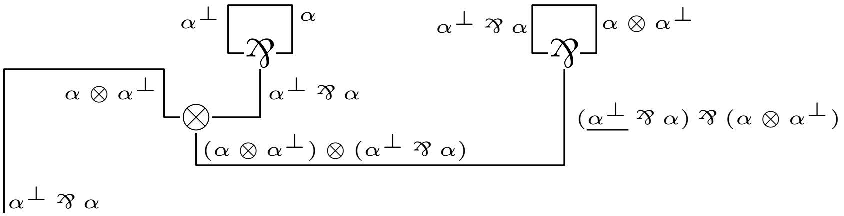
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\parr$
$\parr$
$\alpha \otimes \alpha^\perp$
$\otimes$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\underline{\alpha} \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

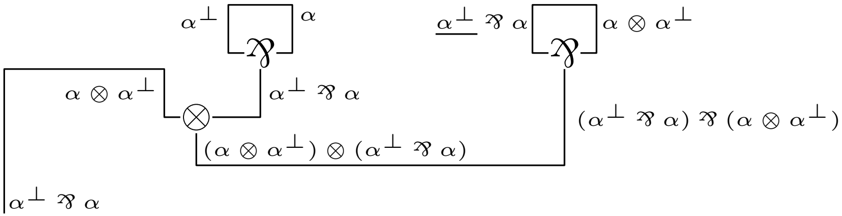
How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$


$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\underline{\alpha^\perp} \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

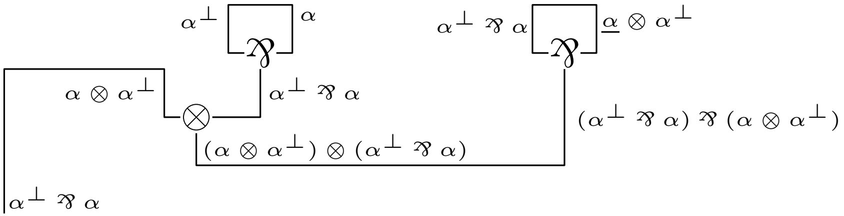
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\underline{\alpha^\perp} \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\parr$
$\parr$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$\otimes$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

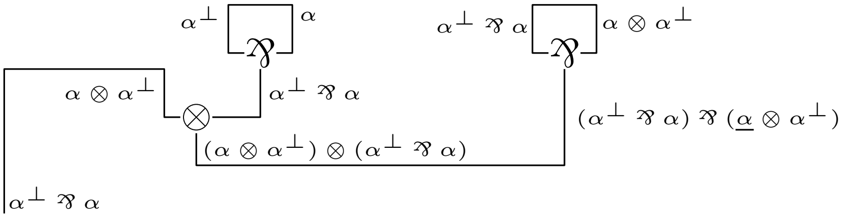
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\underline{\alpha} \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

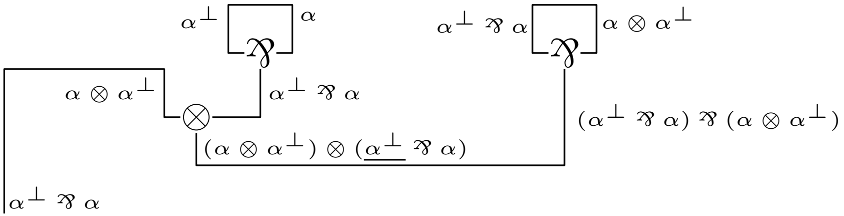
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\underline{\alpha} \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

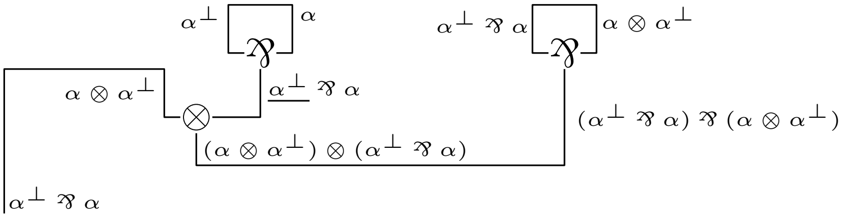
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\underline{\alpha^\perp} \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

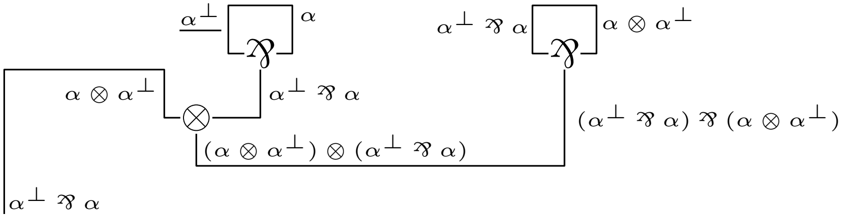
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\underline{\alpha^\perp} \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

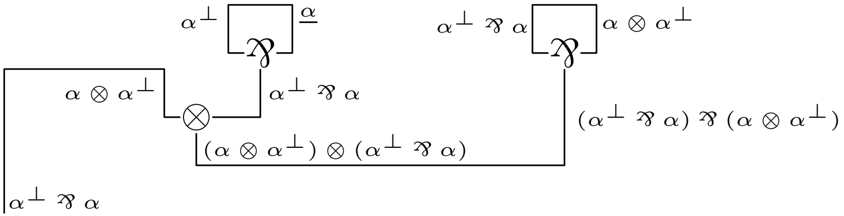
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

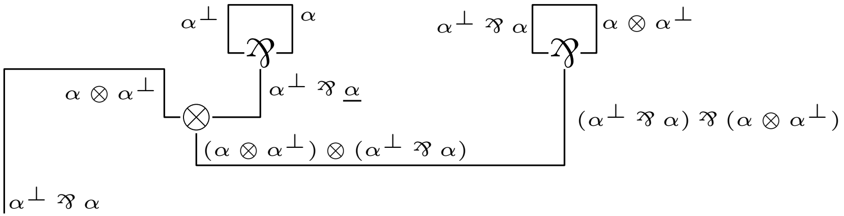
How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$


$\alpha^\perp$
$\underline{\alpha}$
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

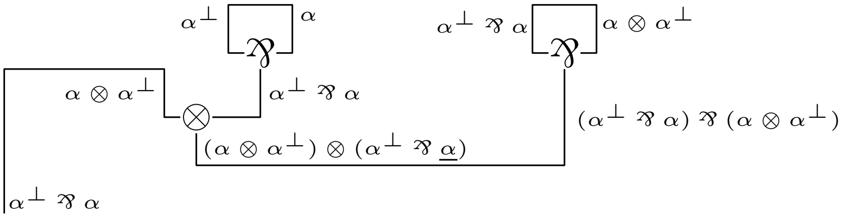
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \underline{\alpha}$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

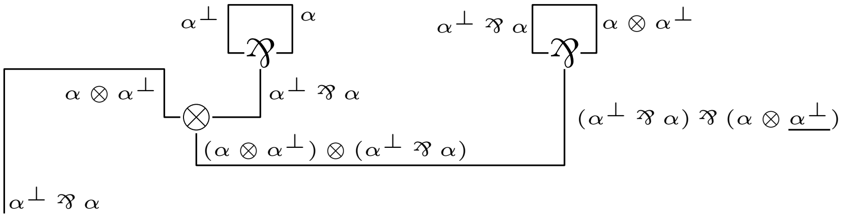
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\parr$
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\parr$
$\alpha \otimes \alpha^\perp$
$\otimes$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \underline{\alpha})$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

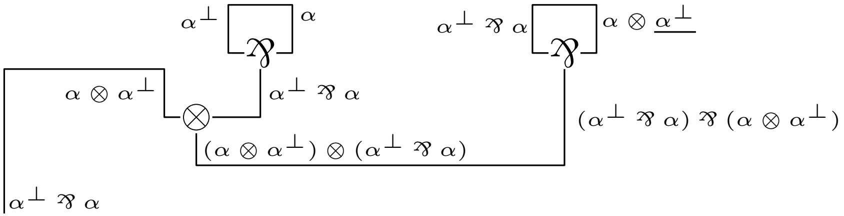
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \underline{\alpha^\perp})$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

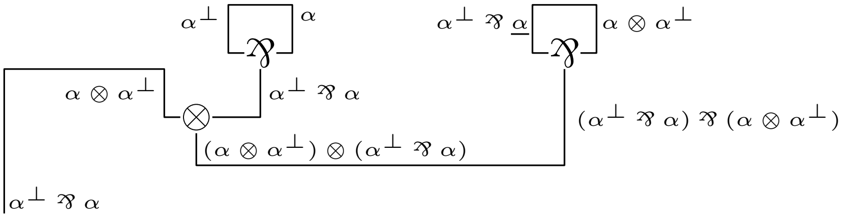
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \underline{\alpha^\perp}$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

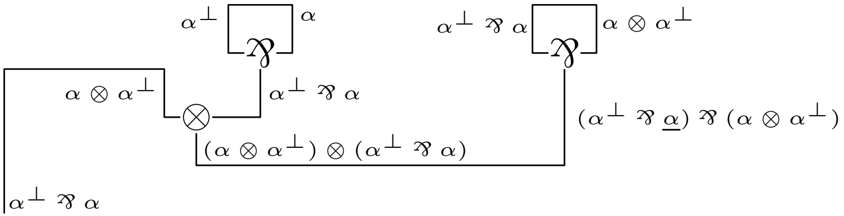
How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$


$\alpha^\perp$
α
$\alpha^\perp \parr \underline{\alpha}$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

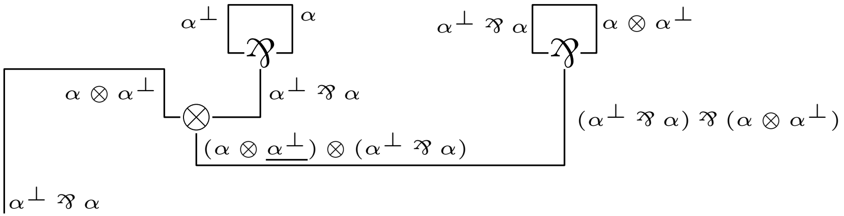
How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$


$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \underline{\alpha}) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

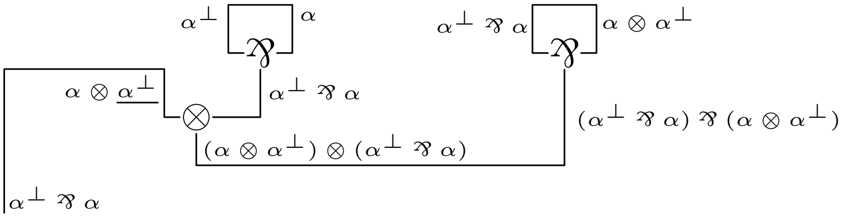
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\parr$
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\parr$
$\alpha \otimes \alpha^\perp$
$\otimes$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \underline{\alpha^\perp}) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

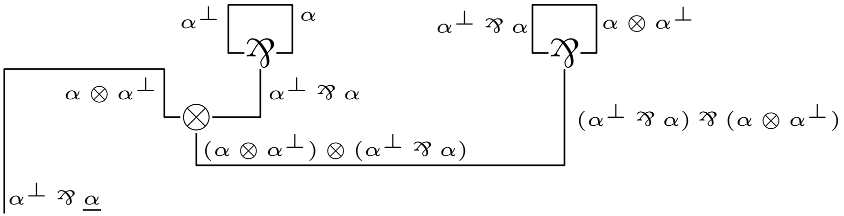
$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \underline{\alpha^\perp}$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \alpha$

How to Capture Persistency? - Example

$$(\lambda x.x)(\lambda x.x)$$



$\alpha^\perp$
α
$\alpha^\perp \parr \alpha$
$\alpha \otimes \alpha^\perp$
$\alpha \otimes \alpha^\perp$
$\alpha^\perp \parr \alpha$
$(\alpha^\perp \parr \alpha) \parr (\alpha \otimes \alpha^\perp)$
$(\alpha \otimes \alpha^\perp) \otimes (\alpha^\perp \parr \alpha)$
$\alpha^\perp \parr \underline{\alpha}$

Persistent Paths as a Model

- Given π, the relation $\rightarrow_\pi$ is deterministic and invertible.
- Are there any (e, C) such that $(e, C) \not\rightarrow_\pi$?
  - If (e, C) does not satisfy the invariant...
  - Or if e is a a conclusion of π and $F(e) = C(\alpha)$ for some atom α.
- Suppose π has just *one* conclusion e.
- Let $\mathcal{C}_\pi^-$ be the set of pairs (e, C) such that:
  - e is the conclusion of π;
  - $F(e) = C(\alpha^\perp)$ for some α.

  Similarly for $\mathcal{C}_\pi^+$.
- **Interpretation**:

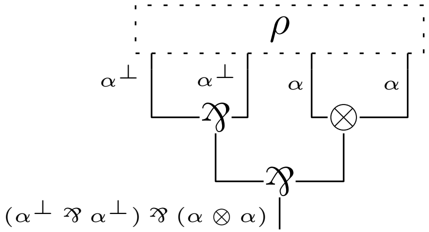
$$[\![\pi]\!] : \mathcal{C}_\pi^- \rightharpoonup \mathcal{C}_\pi^+$$

Proposition (Soundness)

For every π, $[\![\pi]\!]$ is total. Moreover, $[\![\pi]\!] = [\![\rho]\!]$ whenever $\pi \rightsquigarrow \rho$.

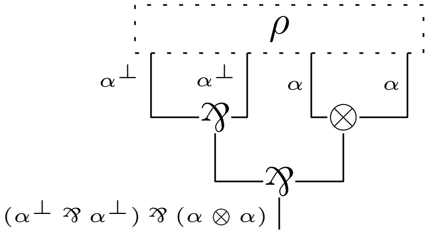
Persistent Paths as a Model

- How much about π can be read from $[\![\pi]\!]$?
- If π is cut-free and axioms are atomic, then π itself can be retrieved from $[\![\pi]\!]$.
- **Example**.
  - Suppose the conclusion of π is $\vdash (\alpha^{\perp} \parr \alpha^{\perp}) \parr (\alpha \otimes \alpha)$.
  - Then π looks as follows:

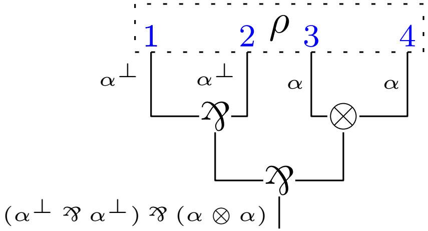


  - ρ is only made of axioms, and can be built by querying $[\![\pi]\!]$ on $(e, ([\cdot] \parr \alpha^{\perp}) \parr (\alpha \otimes \alpha))$ and $(e, (\alpha^{\perp} \parr [\cdot]) \parr (\alpha \otimes \alpha))$

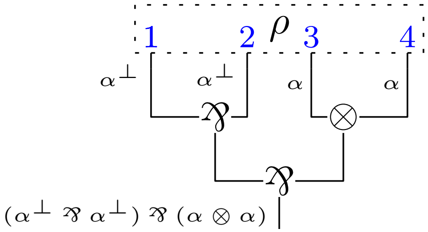
Persistent Paths as a Model



Persistent Paths as a Model



Persistent Paths as a Model



$$\rho_{\mathsf{true}} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Digression: More on Matrices

$\pi : \vdash \Gamma, A$ $\qquad\qquad$ $\sigma : \vdash A^\perp, \Delta$

ρ_π $\qquad\qquad$ ρ_σ

$$\rho_\pi = \left[\begin{array}{c|c} \pi^\Gamma_\Gamma & \pi^\Gamma_A \\ \hline \pi^A_\Gamma & \pi^A_A \end{array}\right] \qquad \rho_\sigma = \left[\begin{array}{c|c} \sigma^{A^\perp}_{A^\perp} & \sigma^{A^\perp}_\Delta \\ \hline \sigma^\Delta_{A^\perp} & \sigma^\Delta_\Delta \end{array}\right]$$

$$\xi = \frac{\pi : \vdash \Gamma, A \quad \sigma : \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

$$\rho_\xi = \left[\begin{array}{c|c} \xi^\Gamma_\Gamma & \xi^\Gamma_\Delta \\ \hline \xi^\Delta_\Gamma & \xi^\Delta_\Delta \end{array}\right]$$

$$= \left[\begin{array}{c|c} \pi^\Gamma_\Gamma + \pi^\Gamma_A \sigma^{A^\perp}_{A^\perp} \sum_{n=0}^\infty (\pi^A_A \sigma^{A^\perp}_{A\perp})^n \pi^A_\Gamma & \pi^\Gamma_A \sum_{n=0}^\infty (\sigma^{A^\perp}_{A^\perp} \pi^A_A)^n \sigma^{A^\perp}_\Delta \\ \hline \sigma^\Delta_{A^\perp} \sum_{n=0}^\infty (\pi^A_A \sigma^{A^\perp}_{A^\perp})^n \pi^A_\Gamma & \rho^{\Delta\Delta}_\rho + \rho^{\Delta A^\perp}_\rho \pi^A_A \sum_{n=0}^\infty (\sigma^{A^\perp}_{A^\perp} \pi^A_A)^n \sigma^{A^\perp}_\Delta \end{array}\right]$$

Digression: More on Matrices

$$\pi : \vdash \Gamma, A \qquad \sigma : \vdash A^\perp, \Delta$$

$$\rho_\pi \qquad \rho_\sigma$$

$$\rho_\pi = \left[\begin{array}{c|c} \pi_\Gamma^\Gamma & \pi_A^\Gamma \\ \hline \pi_\Gamma^A & \pi_A^A \end{array}\right] \qquad \rho_\sigma = \left[\begin{array}{c|c} \sigma_{A^\perp}^{A^\perp} & \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta & \sigma_\Delta^\Delta \end{array}\right]$$

$$\xi = \frac{\pi : \vdash \Gamma, A \quad \sigma : \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

$$\rho_\xi = \left[\begin{array}{c|c} \xi_\Gamma^\Gamma & \xi_\Delta^\Gamma \\ \hline \xi_\Gamma^\Delta & \xi_\Delta^\Delta \end{array}\right]$$

$$= \left[\begin{array}{c|c} \pi_\Gamma^\Gamma + \pi_A^\Gamma \sigma_{A^\perp}^{A^\perp} \sum_{n=0}^\infty (\pi_A^A \sigma_{A\perp}^{A^\perp})^n \pi_\Gamma^A & \pi_A^\Gamma \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \rho_\rho^{\Delta\Delta} + \rho_\rho^{\Delta A^\perp} \pi_A^A \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \end{array}\right]$$

Digression: More on Matrices

$$\pi : \vdash \Gamma, A \qquad\qquad \sigma : \vdash A^\perp, \Delta$$

$$\rho_\pi \qquad\qquad \rho_\sigma$$

$$\rho_\pi = \left[\begin{array}{c|c} \pi_\Gamma^\Gamma & \pi_A^\Gamma \\ \hline \pi_\Gamma^A & \pi_A^A \end{array}\right] \qquad\qquad \rho_\sigma = \left[\begin{array}{c|c} \sigma_{A^\perp}^{A^\perp} & \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta & \sigma_\Delta^\Delta \end{array}\right]$$

$$\xi = \frac{\pi : \vdash \Gamma, A \quad \sigma : \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

$$\rho_\xi = \left[\begin{array}{c|c} \xi_\Gamma^\Gamma & \xi_\Delta^\Gamma \\ \hline \xi_\Gamma^\Delta & \xi_\Delta^\Delta \end{array}\right]$$

$$= \left[\begin{array}{c|c} \pi_\Gamma^\Gamma + \pi_A^\Gamma \sigma_{A^\perp}^{A^\perp} \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \pi_A^\Gamma \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \rho_\rho^{\Delta\Delta} + \rho_\rho^{\Delta A^\perp} \pi_A^A \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \end{array}\right]$$

Digression: More on Matrices

$$\pi : \vdash \Gamma, A \qquad\qquad \sigma : \vdash A^\perp, \Delta$$

$$\rho_\pi \qquad\qquad \rho_\sigma$$

$$\rho_\pi = \left[\begin{array}{c|c} \pi_\Gamma^\Gamma & \pi_A^\Gamma \\ \hline \pi_\Gamma^A & \pi_A^A \end{array}\right] \qquad \rho_\sigma = \left[\begin{array}{c|c} \sigma_{A^\perp}^{A^\perp} & \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta & \sigma_\Delta^\Delta \end{array}\right]$$

$$\xi = \frac{\pi : \vdash \Gamma, A \quad \sigma : \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

$$\rho_\xi = \left[\begin{array}{c|c} \xi_\Gamma^\Gamma & \xi_\Delta^\Gamma \\ \hline \xi_\Gamma^\Delta & \xi_\Delta^\Delta \end{array}\right]$$

$$= \left[\begin{array}{c|c} \pi_\Gamma^\Gamma + \pi_A^\Gamma \sigma_{A^\perp}^{A^\perp} \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \pi_A^\Gamma \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \rho_\rho^{\Delta\Delta} + \rho_\rho^{\Delta A^\perp} \pi_A^A \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \end{array}\right]$$

Digression: More on Matrices

$\pi : \vdash \Gamma, A$ $\qquad$ $\sigma : \vdash A^\perp, \Delta$

ρ_π $\qquad$ ρ_σ

$$\rho_\pi = \left[\begin{array}{c|c} \pi_\Gamma^\Gamma & \pi_A^\Gamma \\ \hline \pi_\Gamma^A & \pi_A^A \end{array}\right] \qquad \rho_\sigma = \left[\begin{array}{c|c} \sigma_{A^\perp}^{A^\perp} & \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta & \sigma_\Delta^\Delta \end{array}\right]$$

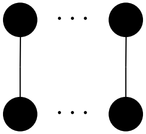
$$\xi = \frac{\pi : \vdash \Gamma, A \quad \sigma : \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

$$\rho_\xi = \left[\begin{array}{c|c} \xi_\Gamma^\Gamma & \xi_\Delta^\Gamma \\ \hline \xi_\Gamma^\Delta & \xi_\Delta^\Delta \end{array}\right]$$

$$= \left[\begin{array}{c|c} \pi_\Gamma^\Gamma + \pi_A^\Gamma \sigma_{A^\perp}^{A^\perp} \sum_{n=0}^\infty (\pi_A^A \sigma_{A\perp}^{A^\perp})^n \pi_\Gamma^A & \pi_A^\Gamma \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \\ \hline \sigma_{A^\perp}^\Delta \sum_{n=0}^\infty (\pi_A^A \sigma_{A^\perp}^{A^\perp})^n \pi_\Gamma^A & \rho_\rho^{\Delta\Delta} + \rho_\rho^{\Delta A^\perp} \pi_A^A \sum_{n=0}^\infty (\sigma_{A^\perp}^{A^\perp} \pi_A^A)^n \sigma_\Delta^{A^\perp} \end{array}\right]$$

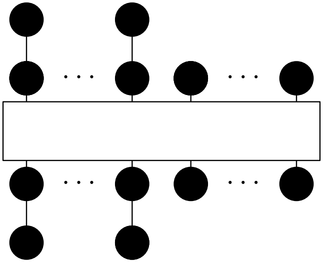
Another Digression: the Structure of Automata

$$\vdash A^{\perp}, A$$



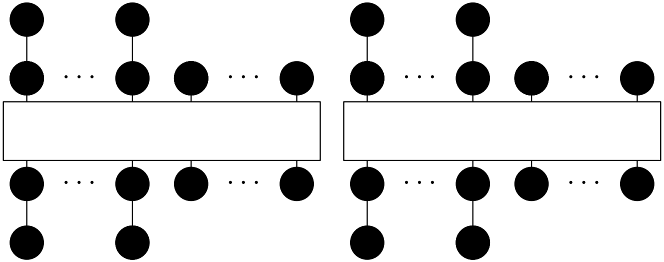
Another Digression: the Structure of Automata

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$



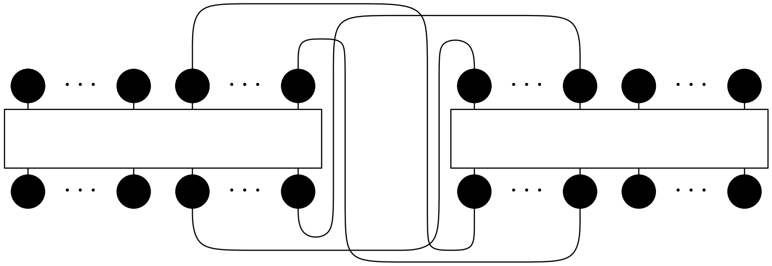
Another Digression: the Structure of Automata

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}$$



Another Digression: the Structure of Automata

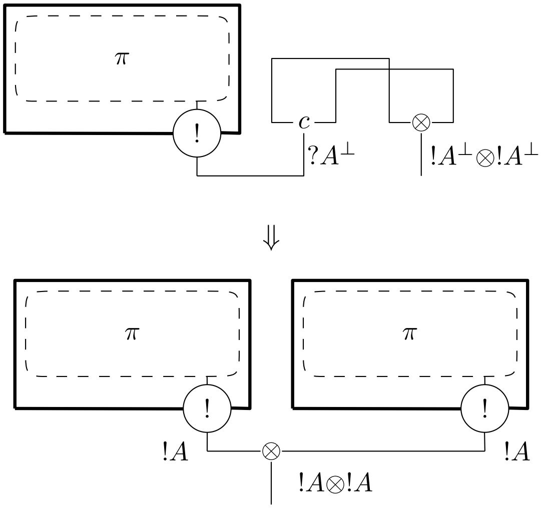
$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}$$



Generalizations

- Geometry of Interaction works reasonably well for systems beyond propositional **MLL**.
- **Exponentials**.
  - States cannot be just pairs (e, C).
  - If $[\cdot]$ appears in the scope of any ! and ? operators in C, then we need to keep track of which particular "copy" of $[\cdot]$ we are talking about.
  - Similarly if e is inside an exponential box.
  - States become tuples in the form (e, C, μ, ν), where μ and ν are sequences of natural numbers.
  - Soundness holds, provided the logical connective ? does not appear in the conclusion of the underlying proof.
- **Second Order Quantification and Recursive Types**.
  - The fundamental invariant does not hold anymore, so C is itself replaced by a string in $\{p, q\}^*$ playing the same role, but having unbounded length.

Generalizations



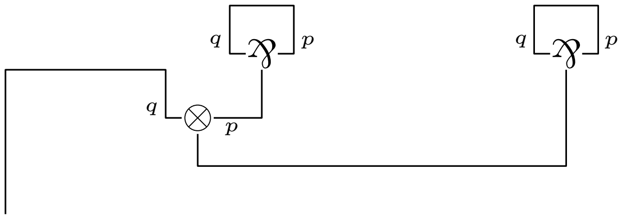
An Algebraic Point of View

- Instead of isolating persistent paths through automata, proceed by assigning to any straight path a weight, and evaluate such a weight using the so-called **path algebra**.
- **Monomials**: p, q, 1, 0.
- **Concatenation of paths**: binary operation $\cdot$, with 1 as an identity and 0 as an absorbing element.
- **Reversing a path**: unary operation $(\cdot)^*$.
- **Equations**:

$$
\begin{aligned}
0^* &= 0 & 1^* &= 1 \\
(x^*)^* &= x & (xy)^* &= y^* x^* \\
q^* q = p^* p &= 1 & q^* p = p^* q &= 0
\end{aligned}
$$

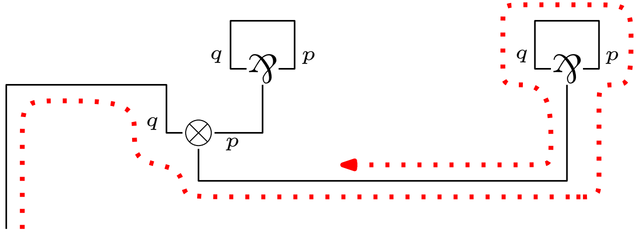
An Algebraic Point of View

$$(\lambda x.x)(\lambda x.x)$$



An Algebraic Point of View

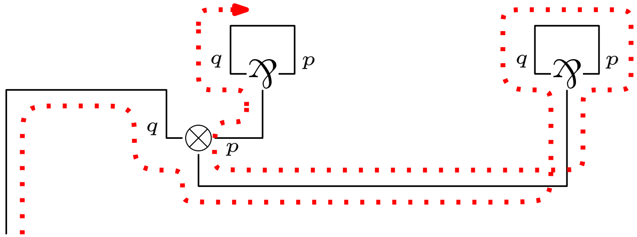
$$(\lambda x.x)(\lambda x.x)$$



$$q^* p q^* = 0 \cdot q^* = 0$$

An Algebraic Point of View

$$(\lambda x.x)(\lambda x.x)$$



$$q^* q p^* p q = 1 \cdot p^* p q = 1 \cdot q = q$$

Applications

- **GoI as a Proof Technique**. Some crucial aspects of the dynamics of cut-elimination are put in evidence by GoI.
- Examples:
  - Correctness of optimal reduction algorithms [GAL1992].
  - Termination of pure nets [DR1993].
- **GoI as an Implementation Technique**. GoI is *effective*. As such, it can be considered itself as a way to compute.
- Examples:
  - Readback algorithms for optimal reduction [GAL1992].
  - (Directed) virtual reduction [DR1992, DPR1993].
  - An interactive machine implementing the λ-calculus [Mackie1994].
  - A parallel machine for the λ-calculus [Pinto1999].

Part II

Applications to ICC

Context: Implicit Complexity

- **Goal**
  - Machine-free characterizations of complexity classes.
  - P, PSPACE, L, NC,...
- **Why?**
  - Simple and elegant presentations of complexity classes.
  - Formal methods for complexity analysis of programs.
- **How?**
  - Recursion theory [BC92], [Leivant94], ...
  - Model theory [Fagin73], ...,
  -

Context: Implicit Complexity

- **Goal**
  - Machine-free characterizations of complexity classes.
  - P, PSPACE, L, NC,...
- **Why?**
  - Simple and elegant presentations of complexity classes.
  - Formal methods for complexity analysis of programs.
- **How?**
  - Recursion theory [BC92], [Leivant94], ...
  - Model theory [Fagin73], ...,
  -

Context: Implicit Complexity

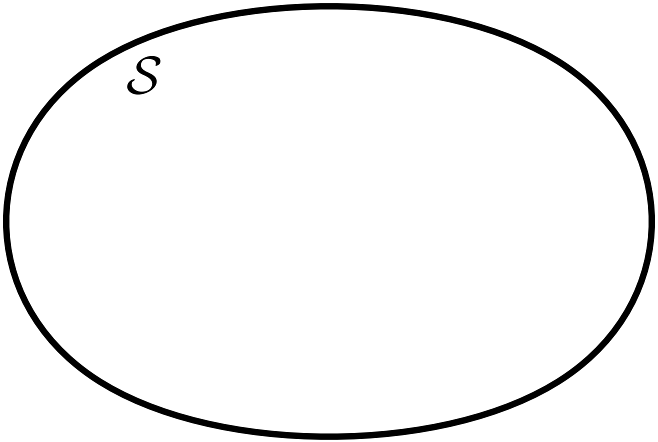
- **Goal**
  - Machine-free characterizations of complexity classes.
  - P, PSPACE, L, NC,...
- **Why?**
  - Simple and elegant presentations of complexity classes.
  - Formal methods for complexity analysis of programs.
- **How?**
  - Recursion theory [BC92], [Leivant94], ...
  - Model theory [Fagin73], ...,
  - Proof theory and λ-calculi

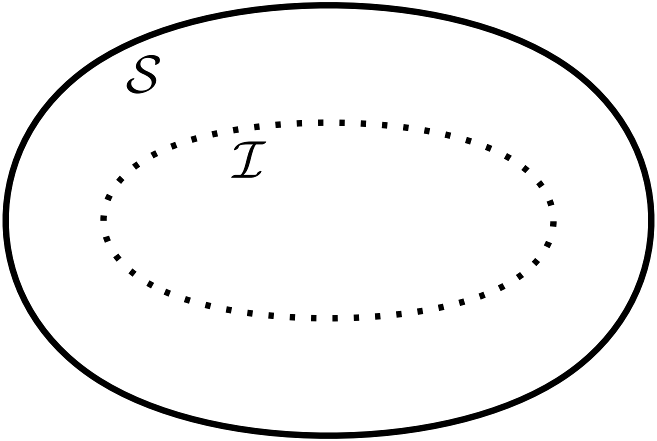
Context: Implicit Complexity

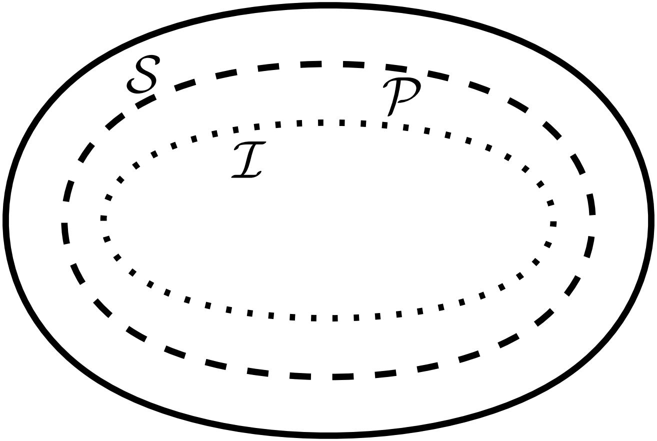
- **Goal**
  - Machine-free characterizations of complexity classes.
  - P, PSPACE, L, NC,...
- **Why?**
  - Simple and elegant presentations of complexity classes.
  - Formal methods for complexity analysis of programs.
- **How?**
  - Recursion theory [BC92], [Leivant94], ...
  - Model theory [Fagin73], ...,
  - Proof theory and λ-calculi

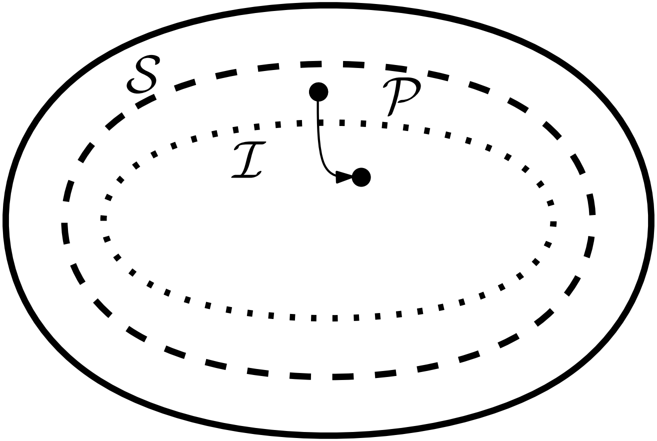
Context: Implicit Complexity

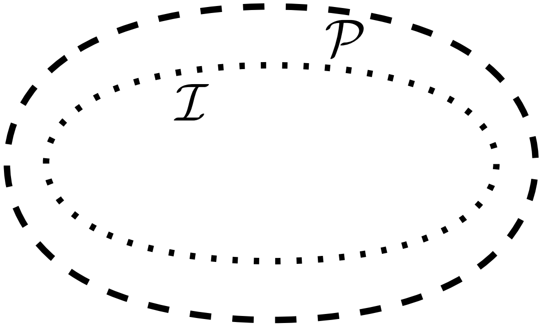
- **Goal**
  - Machine-free characterizations of complexity classes.
  - P, PSPACE, L, NC,...
- **Why?**
  - Simple and elegant presentations of complexity classes.
  - Formal methods for complexity analysis of programs.
- **How?**
  - Recursion theory [BC92], [Leivant94], ...
  - Model theory [Fagin73], ...,
  - Proof theory and λ-calculi ... by way of “linear techniques”.


$\mathcal{S}$


$\mathcal{S}$
$\mathcal{I}$


$\mathcal{S}$
$\mathcal{P}$
$\mathcal{I}$


$\mathcal{S}$
$\mathcal{P}$
$\mathcal{I}$


$\mathcal{P}$
$\mathcal{I}$

Hot to Prove $\mathcal{I} \subseteq \mathcal{P}$?

- This corresponds to **Soundness** wrt a complexity class.
- **Natural** solution: analyze the combinatorics of $\mathcal{I}$.
  - Studying cut-elimination, normalization, evaluation, etc.
  - **Apparently**, this is the simplest solution.
- What if $\mathcal{I}_1, \ldots, \mathcal{I}_n \subseteq \mathcal{P}$?
  - The proofs would be similar;
  - But more or less everything must be redone;
  - Even worse when $\mathcal{I}_1 \subseteq \mathcal{P}_1, \ldots, \mathcal{I}_n \subseteq \mathcal{P}_n$

Factorizing Through $\mathbf{W}(\cdot)$

$$\pi \in \mathcal{S} \longmapsto \mathbf{W}(\pi) \in \mathbb{N}$$

$$\forall \pi \in \mathcal{S} \qquad \mathbf{W}(\pi) \sim \mathit{Complexity}(\pi)$$

- $\mathbf{W}(\cdot)$ should be easier to compute (and reason about) than $\mathit{Complexity}(\cdot)$ itself!
- $\mathbf{W}(\pi)$ needs to reveal something about the dynamics of π;
- $\mathbf{W}(\pi)$ can be "read" from $[\![\pi]\!]$.

Factorizing Through $\mathbf{W}(\cdot)$

$$\pi \in \mathcal{S} \longmapsto \mathbf{W}(\pi) \in \mathbb{N}$$

$$\forall \pi \in \mathcal{S} \qquad \mathbf{W}(\pi) \sim \mathit{Complexity}(\pi)$$

- $\mathbf{W}(\cdot)$ should be easier to compute (and reason about) than $\mathit{Complexity}(\cdot)$ itself!
- $\mathbf{W}(\pi)$ needs to reveal something about the dynamics of π;
- $\mathbf{W}(\pi)$ can be "read" from $[\![\pi]\!]$.

Factorizing Through $\mathbf{W}(\cdot)$

$$\pi \in \mathcal{S} \longmapsto \mathbf{W}(\pi) \in \mathbb{N}$$

$$\forall \pi \in \mathcal{S} \qquad \mathbf{W}(\pi) \sim \mathit{Complexity}(\pi)$$

- $\mathbf{W}(\cdot)$ should be easier to compute (and reason about) than $\mathit{Complexity}(\cdot)$ itself!
- $\mathbf{W}(\pi)$ needs to reveal something about the dynamics of π;
- $\mathbf{W}(\pi)$ can be "read" from $[\![\pi]\!]$.

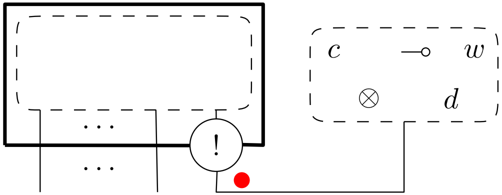
A Case Study: $\mathcal{S} \equiv$ MELL

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
  - ... in any strategy.

Proposition

$\mathbf{W}(\pi)$ and $Time(\pi)$ are related by polynomials.

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



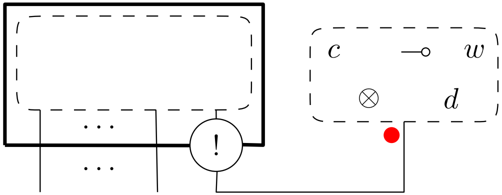
A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
    - ... in any strategy.

Proposition

$\mathbf{W}(\pi)$ *and* $Time(\pi)$ *are related by polynomials.*

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



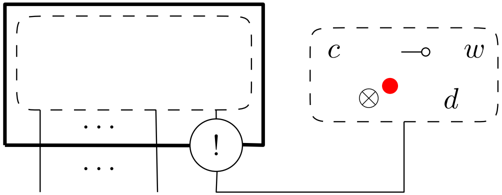
A Case Study: $\mathcal{S} \equiv$ MELL

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
  - ... in any strategy.

Proposition

$\mathbf{W}(\pi)$ and $Time(\pi)$ are related by polynomials.

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



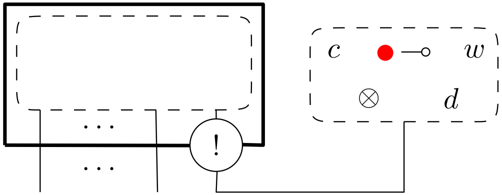
A Case Study: $\mathcal{S} \equiv$ MELL

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
    - ... in any strategy.

Proposition

$\mathbf{W}(\pi)$ and $Time(\pi)$ are related by polynomials.

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



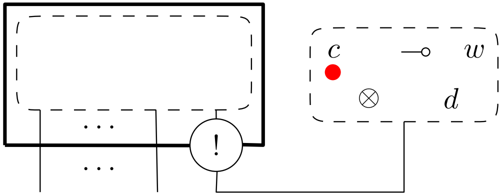
A Case Study: $\mathcal{S} \equiv$ MELL

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
  - ... in any strategy.

Proposition

$\mathbf{W}(\pi)$ *and* $Time(\pi)$ *are related by polynomials.*

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



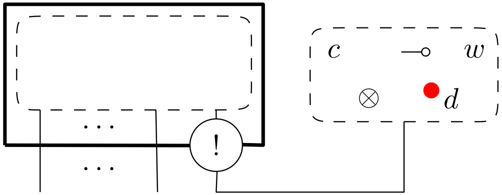
A Case Study: $\mathcal{S} \equiv$ MELL

- Let $\mathbf{W}(\pi)$ be the number of times boxes are copied along the normalization of π...
  - ... in any strategy.

Proposition

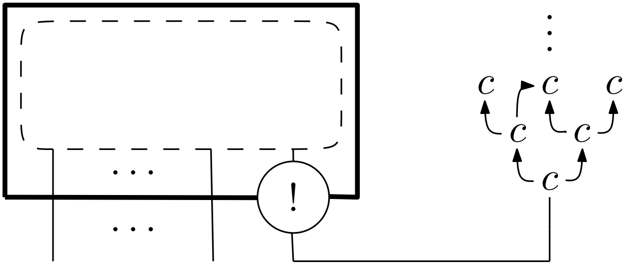
$\mathbf{W}(\pi)$ and $Time(\pi)$ are related by polynomials.

- $\mathbf{W}(\pi)$ can be computed from the *GoI interpretation* $[\![\pi]\!]$ of π:



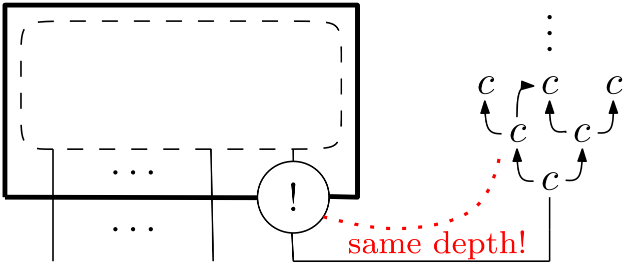
A Case Study: $\mathcal{S} \equiv$ MELL

$$\mathcal{I} \equiv \mathsf{ELL}$$



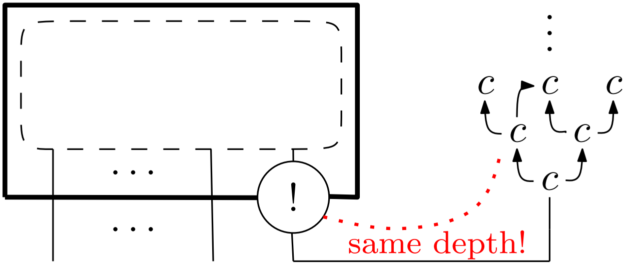
A Case Study: $\mathcal{S} \equiv$ MELL

$\mathcal{I} \equiv$ ELL



A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

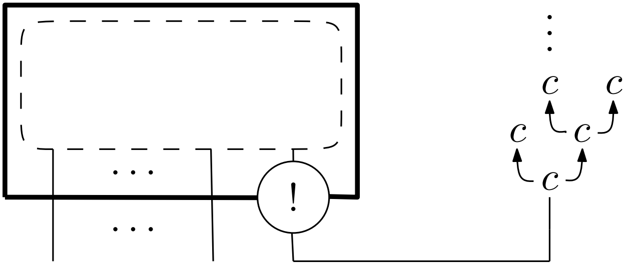
$\mathcal{I} \equiv \mathsf{ELL}$



$\mathsf{ELL} \subseteq \mathsf{ELTIME}$

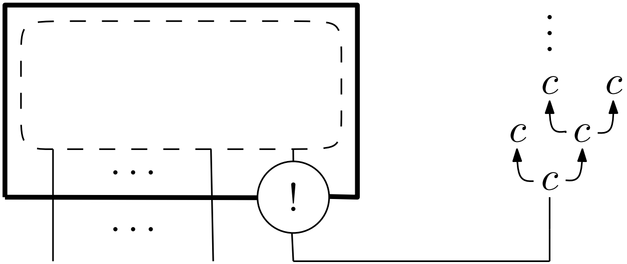
A Case Study: $\mathcal{S} \equiv$ MELL

$$\mathcal{I} \equiv \mathsf{LLL}$$



A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

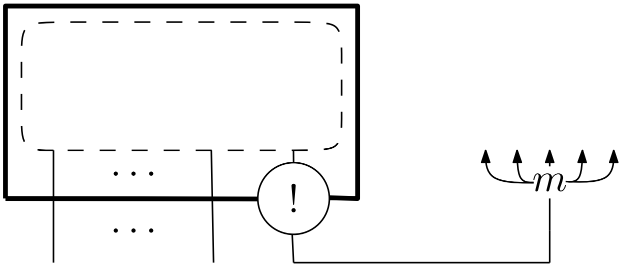
$$\mathcal{I} \equiv \mathsf{LLL}$$



$$\mathsf{LLL} \subseteq \mathsf{FPTIME}$$

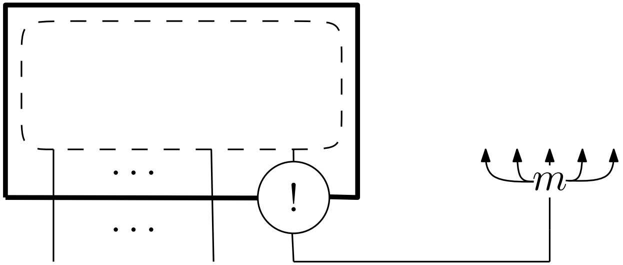
A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

$\mathcal{I} \equiv \mathsf{SLL}$



A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

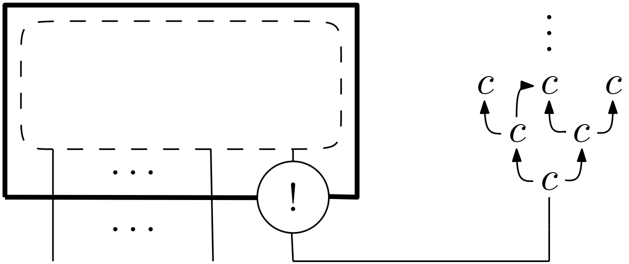
$$\mathcal{I} \equiv \mathsf{SLL}$$



$$\mathsf{SLL} \subseteq \mathsf{FPTIME}$$

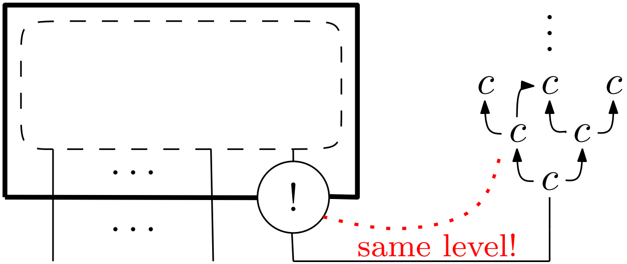
A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

$$\mathcal{I} \equiv \mathsf{L}^3$$



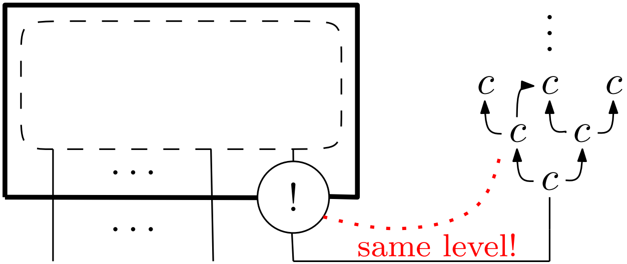
A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

$$\mathcal{I} \equiv \mathsf{L}^3$$



A Case Study: $\mathcal{S} \equiv \mathsf{MELL}$

$$\mathcal{I} \equiv \mathsf{L}^3$$



$$\mathsf{L}^3 \subseteq \mathsf{ELTIME}$$

Same Idea in Other Contexts

- **Linear Higher-Order Recursion**.
  - Gödel's $\mathcal{T}$ when contraction is restricted to base types.
  - Possibly endowed with ramification conditions [Leivant1994,Hofmann1997]
  - $\mathbf{W}(M)$ is the maximum size of *first-order terms* appearing along a reduction sequence for M.
  - Results:

	T	A	W	∅
H(·)	PA	PR	PR	PR
RH(·)	E	E	P	P

- **Optimal Reduction**.
  - Interaction nets as a way to implement λ-calculus optimal reduction.
  - $\mathbf{W}(G)$ is the total number of times *fan-in* and *fan-out* nodes are duplicated along the reduction of the graph G.
  - Results: a study of optimal reduction *actual* performance when done on terms coming from ELL or LLL.

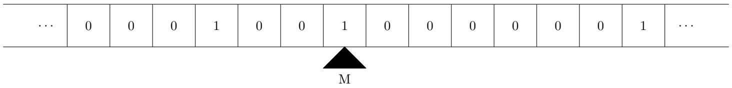
Another Application: Sublinear Space Computation

- Computation with data **too large** to fit into memory.
  - Input is accessed interactively, piece by piece, with random access.
  - Output can only be produced interactively.
- Complexity classes which fit in this scenario: L, NL, etc.
- How to write programs working in sublinear space?
  - Cannot store intermediate values when composing programs...
  - The same computation is possibly performed repeatedly.
- Is there any **natural** characterization of the sublinear space classes in terms of higher-order programming languages?
  - We would like a programming language enjoying "closure properties" similar to the one of the underlying complexity class.

Another Application: Sublinear Space Computation

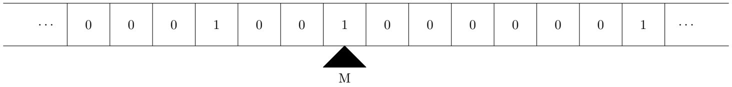
- Computation with data **too large** to fit into memory.
  - Input is accessed interactively, piece by piece, with random access.
  - Output can only be produced interactively.
- Complexity classes which fit in this scenario: L, NL, etc.
- How to write programs working in sublinear space?
  - Cannot store intermediate values when composing programs...
  - The same computation is possibly performed repeatedly.
- Is there any **natural** characterization of the sublinear space classes in terms of higher-order programming languages?
  - We would like a programming language enjoying "closure properties" similar to the one of the underlying complexity class.

From Ordinary Turing Machines...



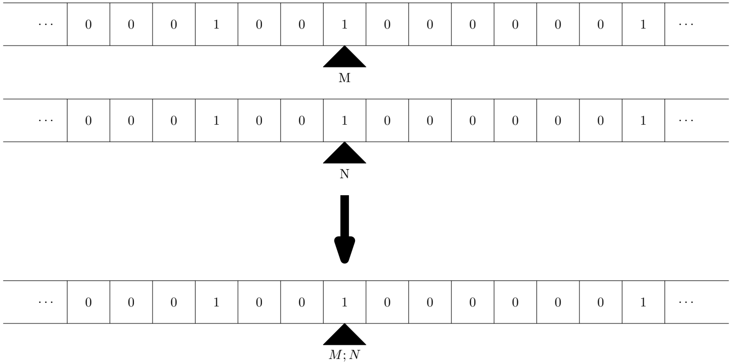
- Time measure: number of transitions.
- Space measure: maximum number of non-blank cells.

From Ordinary Turing Machines...



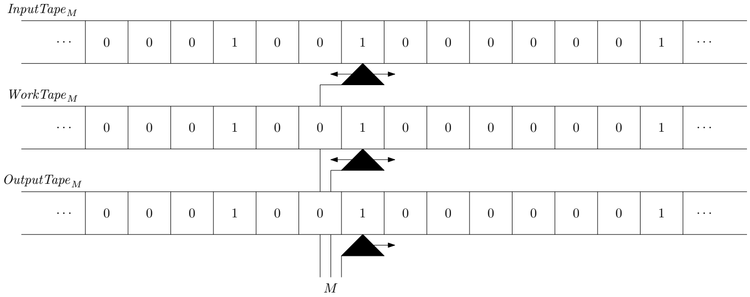
- Time measure: number of transitions.
- Space measure: maximum number of non-blank cells.

From Ordinary Turing Machines...



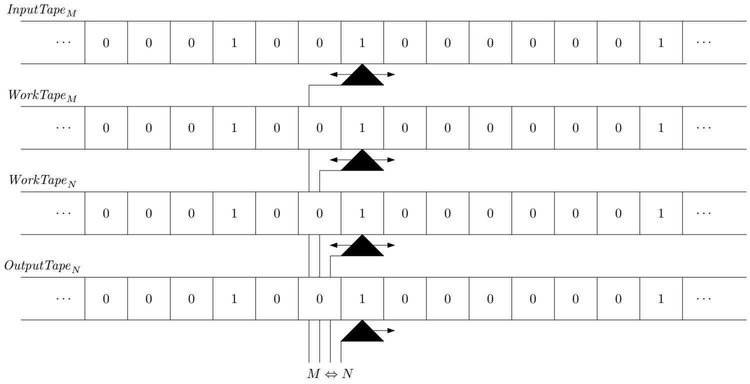
$M; N$ is the batch composition of M and N

… to Offline Turing Machines

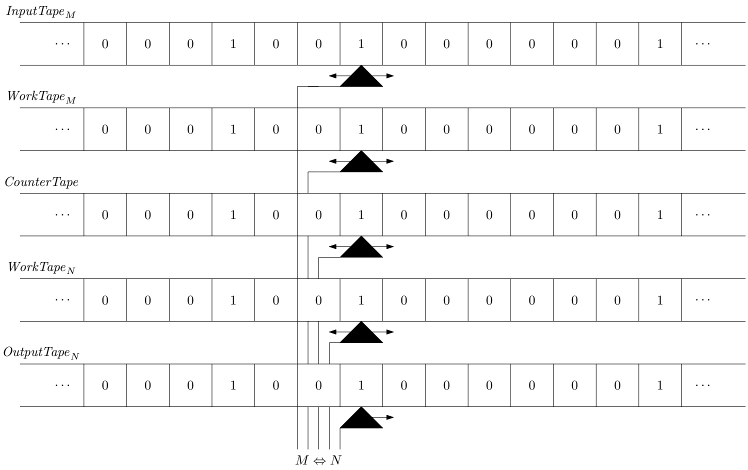


- Time measure: number of transitions.
- Space measure: only the work tape counts.

... to Offline Turing Machines



… to Offline Turing Machines



$M \Leftrightarrow N$ is the interactive composition of M and N

What About Functional Programming?

- Ordinary evaluation mechanisms are inherently non-interactive.
- Composition in the λ-calculus:

$$M, N \Rightarrow \lambda x.M(Nx)$$

- Space measure: size of intermediate values.
- CbV is not space-efficient:

$$(\lambda x.M(Nx))V \to M(NV) \to^* MW \to^* Z$$

Intermediate value W appears explicitly during the computation

- **Goal**: "offline" λ-calculus...

What About Functional Programming?

- Ordinary evaluation mechanisms are inherently non-interactive.
- Composition in the λ-calculus:

$$M, N \Rightarrow \lambda x.M(Nx)$$

- Space measure: size of intermediate values.
- CbV is not space-efficient:

$$(\lambda x.M(Nx))V \to M(NV) \to^* MW \to^* Z$$

Intermediate value W appears explicitly during the computation

- **Goal**: "offline" λ-calculus...

What About Functional Programming?

- Ordinary evaluation mechanisms are inherently non-interactive.
- Composition in the λ-calculus:

$$M, N \Rightarrow \lambda x.M(Nx)$$

- Space measure: size of intermediate values.
- CbV is not space-efficient:

$$(\lambda x.M(Nx))V \to M(NV) \to^* MW \to^* Z$$

Intermediate value W appears explicitly during the computation

- **Goal**: “offline” λ-calculus…

What About Functional Programming?

- Ordinary evaluation mechanisms are inherently non-interactive.
- Composition in the λ-calculus:

$$M, N \Rightarrow \lambda x.M(Nx)$$

- Space measure: size of intermediate values.
- CbV is not space-efficient:

$$(\lambda x.M(Nx))V \to M(NV) \to^* MW \to^* Z$$

Intermediate value W appears explicitly during the computation

- **Goal**: “offline” λ-calculus…

What About Functional Programming?

- Ordinary evaluation mechanisms are inherently non-interactive.
- Composition in the λ-calculus:

$$M, N \Rightarrow \lambda x.M(Nx)$$

- Space measure: size of intermediate values.
- CbV is not space-efficient:

$$(\lambda x.M(Nx))V \to M(NV) \to^* MW \to^* Z$$

  Intermediate value W appears explicitly during the computation
- **Goal**: "offline" λ-calculus...

Main Ideas

- **Keep** the λ-calculus as the underlying programming language.
- **Compiling** every λ-term M to an equivalent, interactive, automaton which computes the GoI interpretation of M.
  - Not really a new idea [Mackie1994], [Pinto2001].
- Space consumption of a program can be **read off** from its type derivation.

Compiling Into an Interactive Form

- Interactive meaning of a type A:

$$A \Longrightarrow (A^-, A^+)$$

  - A^-: questions for A;
  - A^+: answers for A.

- Interactive meaning of a program M:

$$M : A \to B$$

$$\Downarrow$$

$$[M] : A^+ + B^- \to A^- + B^+$$

Compiling Into an Interactive Form

- Interactive meaning of a type A:

$$A \Longrightarrow (A^-, A^+)$$

  - A^-: questions for A;
  - A^+: answers for A.

- Interactive meaning of a program M:

$$M : A \to B$$

$$\Downarrow$$

$$[M] : A^+ + B^- \to A^- + B^+$$

Compiling Into an Interactive Form

- Interactive meaning of a type A:

$$A \Longrightarrow (A^-, A^+)$$

  - A^-: questions for A;
  - A^+: answers for A.

- Interactive meaning of a program M:

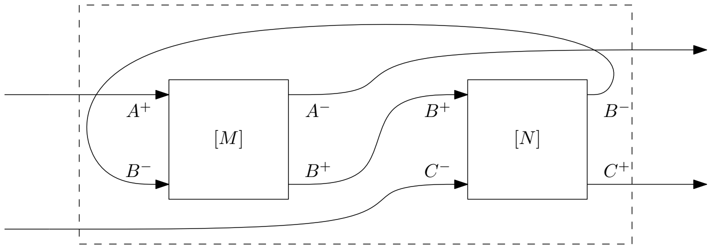
$$M : A \to B$$

$$\Downarrow$$

$$[M] : A^+ + B^- \to A^- + B^+$$

Compiling Into an Interactive Form

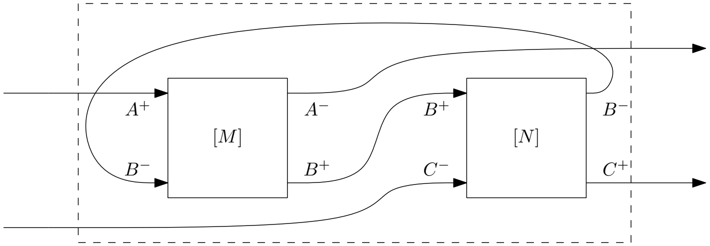
- Primitives (combinators): easy.
- Composition ($M : A \to B$ and $N : B \to C$):



- $[M]$ is an automaton computing the interpretation of M.
- $[M]$ can be seen as a message passing network.

Compiling Into an Interactive Form

- Primitives (combinators): easy.
- Composition ($M : A \to B$ and $N : B \to C$):



- $[M]$ is an automaton computing the interpretation of M.
- $[M]$ can be seen as a message passing network.

A Simple, First-Order, Functional Language O

- Finite Types:

$$A ::= \alpha \mid 1 \mid A + A \mid A \times A$$

- Ordering on all types:

$$\mathtt{min}_A \mid \mathtt{succ}_A(M) \mid \mathtt{eq}_A(M, N)$$

- Loops:

$$\mathtt{loop}(c.M)(N)$$

- CbV evaluation.
  - Harmless: this is the language in which we write automata.
- The **object** language.
- The space consumption of $t : A \to B$ is proportional to the "size" of its type.

Towards IntML

- Syntactically:
  - Enrich the language with higher-order types:

$$X ::= [A] \mid X \otimes X \mid A \cdot X \multimap X$$

  - A linear lambda calculus with pairs.
  - Terms from O appears inlined, e.g. $[M]$.
  - All computation is done by O.
- Semantically:
  - Take any model (e.g. the term model) $\mathbb{O}$ of O.
  - Apply the Int-construction [JSV96] to it, obtaining $Int(\mathbb{O})$.
  - $Int(\mathbb{O})$ is a model of IntML “for free”.

Towards IntML

- Syntactically:
  - Enrich the language with higher-order types:

$$X ::= [A] \mid X \otimes X \mid A \cdot X \multimap X$$

  - A linear lambda calculus with pairs.
  - Terms from O appears inlined, e.g. $[M]$.
  - All computation is done by O.
- Semantically:
  - Take any model (e.g. the term model) $\mathbb{O}$ of O.
  - Apply the Int-construction [JSV96] to it, obtaining $Int(\mathbb{O})$.
  - $Int(\mathbb{O})$ is a model of IntML “for free”.

Why Sublinear Space?

Theorem

Any O *term* $M : A \to B$ *can be evaluated on any input* $c : A$ *in space proportional to* $|c|$.

- Why sublinear, then?
- Interaction allows for an exponential improvement:

Strings	$S_\alpha = [\alpha] \multimap [3]$
Graphs	$G_\alpha = ([\alpha] \multimap [2]) \otimes ([\alpha \times \alpha] \multimap [2])$

Theorem

Any IntML *term* $t : S_\alpha \multimap S_{P(\alpha)}$ *computes a logspace function.*

- Also the converse holds:

Theorem

For every logspace function, there is an IntML *term* $M : S_\alpha \multimap S_{P(\alpha)}$ *which computes it.*

Why Sublinear Space?

Theorem

Any O *term* $M : A \to B$ *can be evaluated on any input* $c : A$ *in space proportional to* $|c|$.

- Why sublinear, then?
- Interaction allows for an exponential improvement:

Strings	$S_\alpha = [\alpha] \multimap [3]$
Graphs	$G_\alpha = ([\alpha] \multimap [2]) \otimes ([\alpha \times \alpha] \multimap [2])$

Theorem

Any IntML *term* $t : S_\alpha \multimap S_{P(\alpha)}$ *computes a logspace function.*

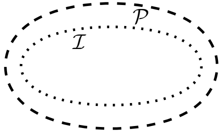
- Also the converse holds:

Theorem

For every logspace function, there is an IntML *term* $M : S_\alpha \multimap S_{P(\alpha)}$ *which computes it.*

A Third Application: ICC and Intensional Expressivity

- ICC systems can be seen as a programming languages guaranteeing quantitative properties of programs:



- Extensional completeness does **not** imply much in terms of intensional expressivity.
  - Can natural algorithms be written in $\mathcal{I}$?
  - Is it possible to design an ICC system such that $\mathcal{I}$ is as close as possible to $\mathcal{P}$?
  - For all "reasonable" complexity classes, $\mathcal{P}$ is not even recursively enumerable...

ICC and Intensional Expressivity

$$\forall \pi \in \mathcal{S} \qquad \mathbf{W}(\pi) \sim \mathit{Complexity}(\pi)$$

ICC and Intensional Expressivity

$$\forall M \in \mathcal{PCF} \qquad \mathbf{W}(M) \sim \mathit{Complexity}(M)$$

ICC and Intensional Expressivity

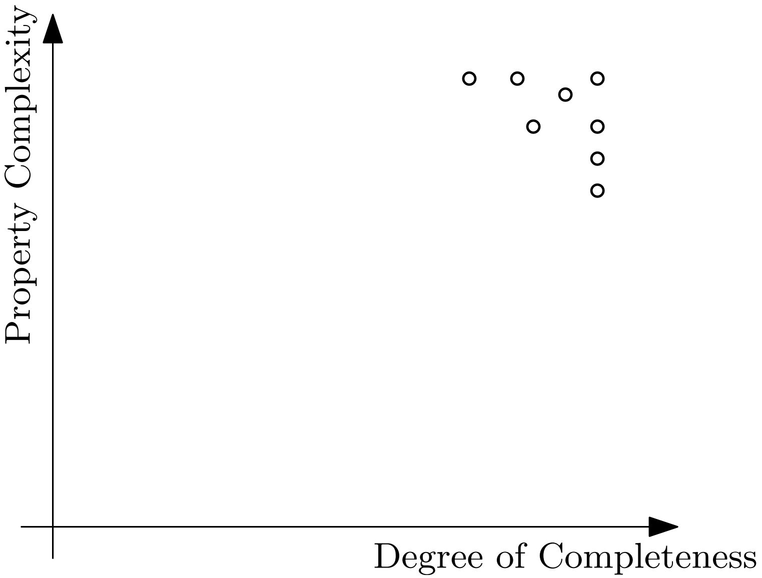
$$\forall M \in \mathcal{PCF} \qquad \mathbf{W}(M) \sim \mathit{Complexity}(M)$$

- **Idea**: internalize the information provided by $\mathbf{W}(\cdot)$ into a type system $\mathcal{T}_{\mathbf{W}}$.
- $\mathbf{W}(\cdot)$ can be read from types. In other words:

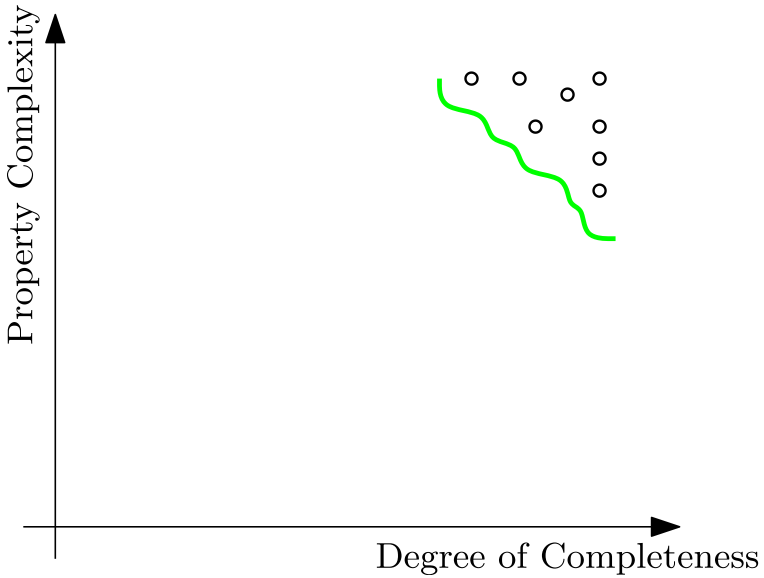
$$\vdash M : A \Leftrightarrow \mathbf{W}(A) = \mathbf{W}(M).$$

- There has to be a price to pay, however.

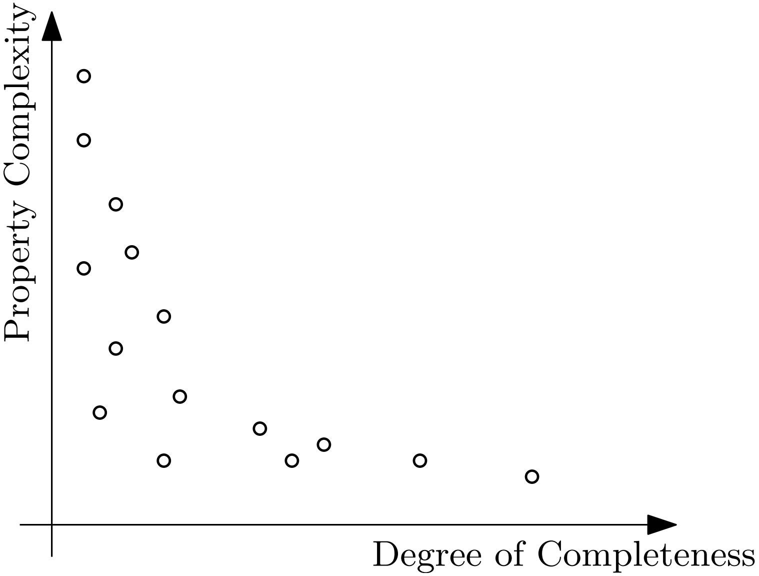
Program Logics



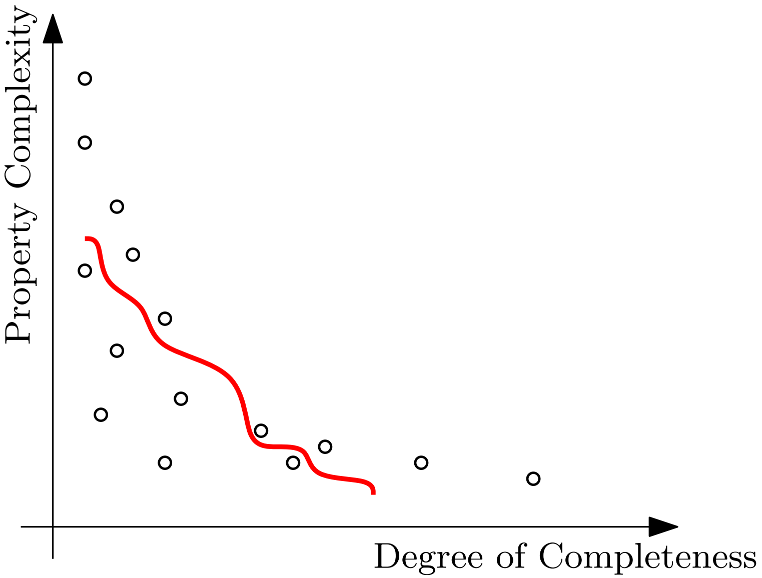
Program Logics



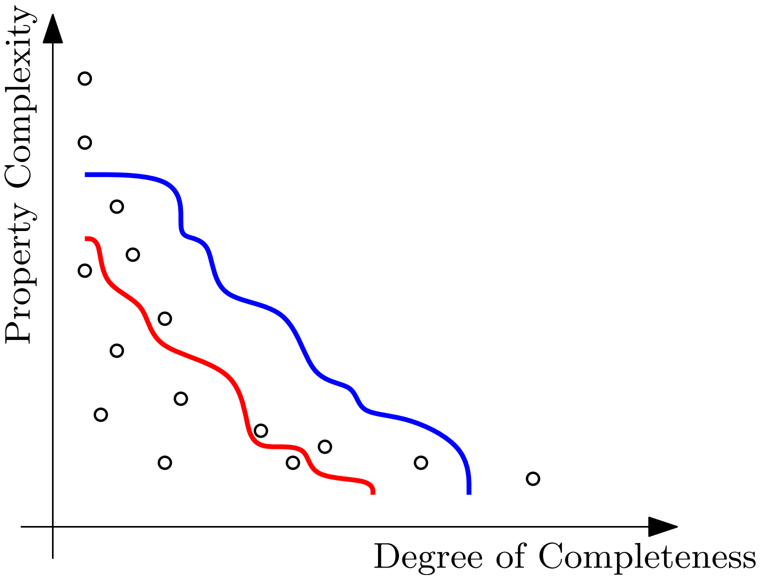
Type Systems



Type Systems



Type Systems



Some Examples

- **Simply Types**
  - "Well-typed programs do not go wrong".
  - Type inference and type checking are often decidable.
- **Dependent Types**
  - Type checking is decidable.
  - Interesting, extensional properties can be specified.
- **Intersection Types**
  - Sound and complete for termination.
  - Type inference is not decidable.
  - Studying programs as *functions* requires considering an **infinite family** of type derivations.

A Notable Exception: Bounded Linear Logic

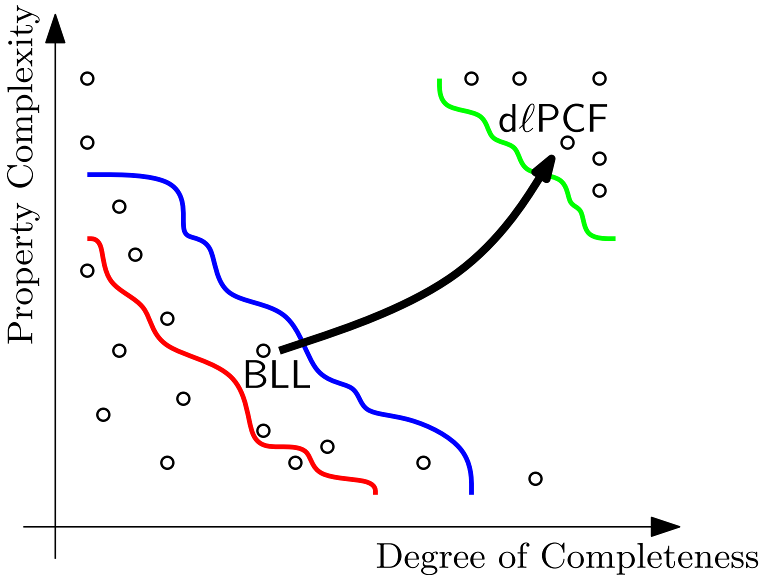
- One of the earliest examples of a system capturing polynomial time **functions** [GSS1992].
  - Extensionally!
  - For every polytime function there is **at least one** proof in BLL computing it.
- Types:

$$A ::= \alpha(p_1, \dots, p_n) \mid A \otimes A \mid A \multimap A \mid \forall \alpha.A \mid !_{x<p}A$$

- How many "polytime proofs" does BLL capture?
  - There's evidence they are **many** [DLHofmann2010].
- Type checking can be **problematic**. As an example:

$$\frac{\Gamma, !_{x<p}A, !_{y<q}A\{p+y/x\} \vdash B \quad p+q \leq r}{\Gamma, !_{x<r}A \vdash B}\ X$$

A Change in Perspective



dℓPCF: a Bird's Eye View

- A type system for the lambda calculus with constants and full higher-order recursion. (i.e. PCF).
- Greatly inspired by BLL.
- Indices are not necessarily polynomials, but terms from a signature Σ.
  - Symbols in Σ are given a meaning by an equational program $\mathcal{E}$.
  - Side conditions in the form:

$$\phi; \Phi \models^{\mathcal{E}} \mathrm{I} \leq \mathrm{J}$$

- Types and modal types are defined as follows:

$$\begin{aligned} A, B &::= \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \mid F \multimap A && \text{basic types} \\ F, G &::= [a < \mathrm{I}] \cdot A && \text{modal types} \end{aligned}$$

dℓPCF: a Bird's Eye View

- A type system for the lambda calculus with constants and full higher-order recursion. (i.e. PCF).
- Greatly inspired by BLL.
- Indices are not necessarily polynomials, but terms from a signature Σ.
  - Symbols in Σ are given a meaning by an equational program $\mathcal{E}$.
  - Side conditions in the form:

$$\phi; \mathrm{K}_1 \leq \mathrm{H}_1, \ldots, \mathrm{K}_n \leq \mathrm{H}_n \models^{\mathcal{E}} \mathrm{I} \leq \mathrm{J}$$

- Types and modal types are defined as follows:

$$\begin{aligned} A, B &::= \mathtt{Nat}[\mathrm{I}, \mathrm{J}] \mid F \multimap A && \text{basic types} \\ F, G &::= [a < \mathrm{I}] \cdot A && \text{modal types} \end{aligned}$$

The Meaning of Types

$$[a < \mathrm{I}] \cdot A \multimap B$$

$$\Downarrow$$

$$(A\{0/a\} \otimes \ldots \otimes A\{\mathrm{I} - 1/a\}) \multimap B$$

The Meaning of Types

$$[a < \mathrm{I}] \cdot A \multimap B$$

$$\Downarrow$$

$$(A\{0/a\} \otimes \ldots \otimes A\{\mathrm{I} - 1/a\}) \multimap B$$

dℓPCF: Intended Meaning

$$a;\emptyset;\emptyset \vdash_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

What does this mean?

- M computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number n, M returns a natural number $\mathrm{K}\{n/a\}$.
- Something more **intensional**:
  - The cost of evaluation of M on an input n is $(\mathrm{I}+\mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a; \emptyset; \emptyset \vdash_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

What does this mean?

- M computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number n, M returns a natural number $\mathrm{K}\{n/a\}$.
- Something more **intensional**:
  - The cost of evaluation of M on an input n is $(\mathrm{I}+\mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a;\emptyset;\emptyset \vdash_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

What does this mean?

- M computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number n, M returns a natural number $\mathrm{K}\{n/a\}$.
- Something more **intensional**:
  - The cost of evaluation of M on an input n is $(\mathrm{I}+\mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

dℓPCF: Intended Meaning

$$a;\emptyset;\emptyset \vdash_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}]$$

What does this mean?

- M computes a function from natural numbers to natural numbers.
- Something **extensional**:
  - On input a natural number n, M returns a natural number $\mathrm{K}\{n/a\}$.
- Something more **intensional**:
  - The cost of evaluation of M on an input n is $(\mathrm{I}+\mathrm{J})\{n/a\}$.
- Two questions:
  - Is this **correct**?
  - How many programs can be captured this way?

Soundness and Completeness

Theorem

Let $\emptyset; \emptyset; \emptyset \vdash_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J}, \mathrm{K}]$ *and* $M \Downarrow^n \underline{\mathtt{m}}$. *Then* $n \leq |M| \cdot [\![\mathrm{I}]\!]^{\mathcal{E}}_{\rho}$

Theorem (Relative Completeness for Programs)

Let M be a PCF *program such that* $M \Downarrow^n \underline{\mathtt{m}}$. *Then, there exist two index terms* I *and* J *such that* $[\![\mathrm{I}]\!]^{\mathcal{U}} \leq n$ *and* $[\![\mathrm{J}]\!]^{\mathcal{U}} = m$ *and such that the term M is typable in* dℓPCF *as*
$\emptyset; \emptyset; \emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J}]$.

Theorem (Relative Completeness for Functions)

Suppose that M is a PCF *term such that* $\vdash M : \mathtt{Nat} \to \mathtt{Nat}$. *Moreover, suppose that there are two (total and computable) functions* $f, g : \mathbb{N} \to \mathbb{N}$ *such that* $M\ \underline{\mathtt{n}} \Downarrow^{g(n)} \underline{\mathtt{f(n)}}$. *Then there are terms* $\mathrm{I}, \mathrm{J}, \mathrm{K}$ *with* $[\![\mathrm{I} + \mathrm{J}]\!] \leq g$ *and* $[\![\mathrm{K}]\!] = f$, *such that*

$$a; \emptyset; \emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}].$$

Soundness and Completeness

Theorem

Let $\emptyset;\emptyset;\emptyset \vdash_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J},\mathrm{K}]$ *and* $M \Downarrow^{n} \underline{\mathtt{m}}$. *Then* $n \leq |M| \cdot [\![\mathrm{I}]\!]^{\mathcal{E}}_{\rho}$

Theorem (Relative Completeness for Programs)

Let M be a PCF *program such that* $M \Downarrow^{n} \underline{\mathtt{m}}$. *Then, there exist two index terms* I *and* J *such that* $[\![\mathrm{I}]\!]^{\mathcal{U}} \leq n$ *and* $[\![\mathrm{J}]\!]^{\mathcal{U}} = m$ *and such that the term M is typable in* dℓPCF *as*
$\emptyset;\emptyset;\emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J}]$.

Theorem (Relative Completeness for Functions)

Suppose that M is a PCF *term such that* $\vdash M : \mathtt{Nat} \to \mathtt{Nat}$. *Moreover, suppose that there are two (total and computable) functions* $f, g : \mathbb{N} \to \mathbb{N}$ *such that* $M\ \underline{\mathtt{n}} \Downarrow^{g(n)} \underline{\mathtt{f(n)}}$. *Then there are terms* $\mathrm{I}, \mathrm{J}, \mathrm{K}$ *with* $[\![\mathrm{I}+\mathrm{J}]\!] \leq g$ *and* $[\![\mathrm{K}]\!] = f$, *such that*

$$a;\emptyset;\emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}].$$

Soundness and Completeness

Theorem

Let $\emptyset;\emptyset;\emptyset \vdash_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J},\mathrm{K}]$ *and* $M \Downarrow^{n} \underline{\mathtt{m}}$*. Then* $n \leq |M| \cdot [\![\mathrm{I}]\!]^{\mathcal{E}}_{\rho}$

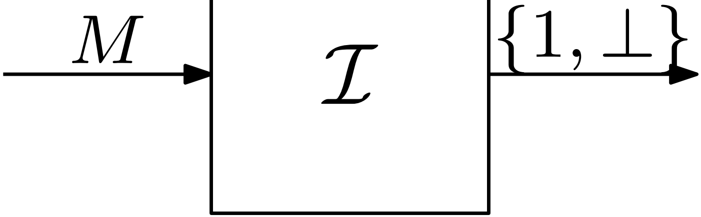
Theorem (Relative Completeness for Programs)

Let M *be a* PCF *program such that* $M \Downarrow^{n} \underline{\mathtt{m}}$*. Then, there exist two index terms* I *and* J *such that* $[\![\mathrm{I}]\!]^{\mathcal{U}} \leq n$ *and* $[\![\mathrm{J}]\!]^{\mathcal{U}} = m$ *and such that the term* M *is typable in* dℓPCF *as*
$\emptyset;\emptyset;\emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : \mathtt{Nat}[\mathrm{J}]$.

Theorem (Relative Completeness for Functions)

Suppose that M *is a* PCF *term such that* $\vdash M : \mathtt{Nat} \to \mathtt{Nat}$*. Moreover, suppose that there are two (total and computable) functions* $f, g : \mathbb{N} \to \mathbb{N}$ *such that* $M\ \underline{\mathtt{n}} \Downarrow^{g(n)} \underline{\mathtt{f(n)}}$*. Then there are terms* $\mathrm{I}, \mathrm{J}, \mathrm{K}$ *with* $[\![\mathrm{I}+\mathrm{J}]\!] \leq g$ *and* $[\![\mathrm{K}]\!] = f$*, such that*

$$a;\emptyset;\emptyset \vdash^{\mathcal{U}}_{\mathrm{I}} M : [b < \mathrm{J}] \cdot \mathtt{Nat}[a] \multimap \mathtt{Nat}[\mathrm{K}].$$


M
$\mathcal{I}$
$\{1, \perp\}$

$$\begin{tikzcd} {} \arrow[r, "M"] & \boxed{d\ell\mathsf{PCF}} \arrow[r, "\Phi"] & {} \end{tikzcd}$$

Thank you!

Questions?