Context Semantics and Implicit Computational Complexity

Ugo Dal Lago

Dipartimento di Informatica
Università di Verona

FOLLIA meeting, January 19th 2006

Outline

Logical Characterization of Complexity Classes

- Various branches of Mathematical Logic have been touched by Implicit Computational Complexity:
  - **Recursion Theory**;
  - **Proof Theory** (Curry-Howard correspondence);
  - **Model Theory** (Finite Model Theory).
- We are mainly interested in proof-theoretical and recursion-theoretical characterizations.
  - They induce programming languages with bounded expressive power.
  - Resource-aware type-systems for existing programming languages can be designed from some of these systems.

Logical Characterization of Complexity Classes

- Various branches of Mathematical Logic have been touched by Implicit Computational Complexity:
  - **Recursion Theory**;
  - **Proof Theory** (Curry-Howard correspondence);
  - **Model Theory** (Finite Model Theory).
- We are mainly interested in proof-theoretical and recursion-theoretical characterizations.
  - They induce programming languages with bounded expressive power.
  - Resource-aware type-systems for existing programming languages can be designed from some of these systems.

Logical Characterization of Complexity Classes

- Various branches of Mathematical Logic have been touched by Implicit Computational Complexity:
  - **Recursion Theory**;
  - **Proof Theory** (Curry-Howard correspondence);
  - **Model Theory** (Finite Model Theory).
- We are mainly interested in proof-theoretical and recursion-theoretical characterizations.
  - They induce programming languages with bounded expressive power.
  - Resource-aware type-systems for existing programming languages can be designed from some of these systems.

Logical Characterization of Complexity Classes

- Various branches of Mathematical Logic have been touched by Implicit Computational Complexity:
  - **Recursion Theory**;
  - **Proof Theory** (Curry-Howard correspondence);
  - **Model Theory** (Finite Model Theory).
- We are mainly interested in proof-theoretical and recursion-theoretical characterizations.
  - They induce programming languages with bounded expressive power.
  - Resource-aware type-systems for existing programming languages can be designed from some of these systems.

Logical Characterization of Complexity Classes

- Various branches of Mathematical Logic have been touched by Implicit Computational Complexity:
  - **Recursion Theory**;
  - **Proof Theory** (Curry-Howard correspondence);
  - **Model Theory** (Finite Model Theory).
- We are mainly interested in proof-theoretical and recursion-theoretical characterizations.
  - They induce programming languages with bounded expressive power.
  - Resource-aware type-systems for existing programming languages can be designed from some of these systems.

The Need for a Unifying Framework

- Many systems are sound and complete w.r.t. relevant complexity classes (polytime functions, elementary functions, etc.).
- Proof techniques for soundness results are ad-hoc and cannot be easily generalized.
- As a consequence:
  - There are many **open problems**. For example: the impact of linearity costraint on the expressive power of higher-order recursion [Hofmann97,BNS00] has not been analyzed precisely.
  - It is not known whether **combining different systems** characterizing the same class would break soundness.
- The above mentioned systems are all extensionally complete but definately **not intensionally complete**.

The Need for a Unifying Framework

- Many systems are sound and complete w.r.t. relevant complexity classes (polytime functions, elementary functions, etc.).
- Proof techniques for soundness results are ad-hoc and cannot be easily generalized.
- As a consequence:
  - There are many **open problems**. For example: the impact of linearity costraint on the expressive power of higher-order recursion [Hofmann97,BNS00] has not been analyzed precisely.
  - It is not known whether **combining different systems** characterizing the same class would break soundness.
- The above mentioned systems are all extensionally complete but definately **not intensionally complete**.

The Need for a Unifying Framework

- Many systems are sound and complete w.r.t. relevant complexity classes (polytime functions, elementary functions, etc.).
- Proof techniques for soundness results are ad-hoc and cannot be easily generalized.
- As a consequence:
  - There are many **open problems**. For example: the impact of linearity costraint on the expressive power of higher-order recursion [Hofmann97,BNS00] has not been analyzed precisely.
  - It is not known whether **combining different systems** characterizing the same class would break soundness.
- The above mentioned systems are all extensionally complete but definately **not intensionally complete**.

The Need for a Unifying Framework

- Many systems are sound and complete w.r.t. relevant complexity classes (polytime functions, elementary functions, etc.).
- Proof techniques for soundness results are ad-hoc and cannot be easily generalized.
- As a consequence:
  - There are many **open problems**. For example: the impact of linearity costraint on the expressive power of higher-order recursion [Hofmann97,BNS00] has not been analyzed precisely.
  - It is not known whether **combining different systems** characterizing the same class would break soundness.
- The above mentioned systems are all extensionally complete but definately **not intensionally complete**.

Context Semantics

- *Intensional* ways of giving semantics to logics and programming languages:
  - **Game semantics** [AJM92];
  - **Geometry of interaction** [Girard89]
  - **Context semantics** [GAL92a,GAL92b].
- Game semantics and geometry of interaction have been already used to study quantitative properties of programs and proofs [BaillotPedicini01,Ghica05].
- We chose context semantics because of its simplicity.
- We applied it to higher-order recursion and to linear logic, obtaining some results.

Context Semantics

- *Intensional* ways of giving semantics to logics and programming languages:
  - **Game semantics** [AJM92];
  - **Geometry of interaction** [Girard89]
  - **Context semantics** [GAL92a,GAL92b].
- Game semantics and geometry of interaction have been already used to study quantitative properties of programs and proofs [BaillotPedicini01,Ghica05].
- We chose context semantics because of its simplicity.
- We applied it to higher-order recursion and to linear logic, obtaining some results.

Context Semantics

- *Intensional* ways of giving semantics to logics and programming languages:
  - **Game semantics** [AJM92];
  - **Geometry of interaction** [Girard89]
  - **Context semantics** [GAL92a,GAL92b].
- Game semantics and geometry of interaction have been already used to study quantitative properties of programs and proofs [BaillotPedicini01,Ghica05].
- We chose context semantics because of its simplicity.
- We applied it to higher-order recursion and to linear logic, obtaining some results.

Characterizations of Complexity Classes by Higher-Order Recursion

- We obtain characterizations of the following complexity classes:
  - PR, the set of (first-order) primitive recursive functions.
  - E, the class of elementary time computable functions.
  - P, the class of polynomial time computable functions.
- Some of them are already known, while other ones are novel.
- The above characterizations are obtained from higher-order (primitive) recursion by fine-tuning two parameters, namely:
  - The class of "contractible" types.
  - Ramification constraints.
- These results appeared in [DalLago05].

Characterizations of Complexity Classes by Higher-Order Recursion

- We obtain characterizations of the following complexity classes:
  - PR, the set of (first-order) primitive recursive functions.
  - E, the class of elementary time computable functions.
  - P, the class of polynomial time computable functions.
- Some of them are already known, while other ones are novel.
- The above characterizations are obtained from higher-order (primitive) recursion by fine-tuning two parameters, namely:
  - The class of "contractible" types.
  - Ramification constraints.
- These results appeared in [DalLago05].

Context Semantics and Linear Logic

From the context semantics of any linear logic proof-net G, a *weight* $W_G \in \mathbb{N} \cup \{\omega\}$ can be defined in such a way that:

Theorem

There is a polynomial $p : \mathbb{N}^2 \to \mathbb{N}$ such that for every proof-net G, G normalizes in at most $p(W_G, |G|)$ steps and the size of any reduct H of G is at most $p(W_G, |G|)$.

Theorem

Let G be a proof-net. If $W_G = \omega$, then G diverges, otherwise there is H with $G \to^{W_G} H$.

Terms

- Let us first consider the sub-calculus handling the algebra $\mathbb{U}$ of unary integers.
- Terms:

$$M ::= x \mid 0 \mid s \mid MM \mid \lambda x.M \mid M \{\!\{M, M\}\!\} \mid M \langle\!\langle M, M\rangle\!\rangle$$

- Reduction rules are call-by-value:

$$(\lambda x.M)\, V \to M\{V/x\}$$

$$0 \langle\!\langle M, N\rangle\!\rangle \to N \qquad 0 \{\!\{M, N\}\!\} \to N$$

$$s\ t \langle\!\langle M, N\rangle\!\rangle \to M\ t\ (t \langle\!\langle M, N\rangle\!\rangle) \qquad s\ t \{\!\{M, N\}\!\} \to M\ t$$

Terms

- Let us first consider the sub-calculus handling the algebra $\mathbb{U}$ of unary integers.
- Terms:

$$M ::= x \mid 0 \mid s \mid MM \mid \lambda x.M \mid M \{\!\{M, M\}\!\} \mid M \langle\!\langle M, M\rangle\!\rangle$$

- Reduction rules are call-by-value:

$$(\lambda x.M)V \to M\{V/x\}$$

$$0 \langle\!\langle M, N\rangle\!\rangle \to N \qquad 0 \{\!\{M, N\}\!\} \to N$$

$$s\ t \langle\!\langle M, N\rangle\!\rangle \to M\ t\ (t \langle\!\langle M, N\rangle\!\rangle) \qquad s\ t \{\!\{M, N\}\!\} \to M\ t$$

Types

- Types:

$$A ::= \mathbb{U}^n \mid A \multimap A$$

- Type-assignment rules:

$$\frac{}{\Gamma, x : A \vdash x : A}\ A \qquad \frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash M\{z/x, z/y\} : B}\ C$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ I_{\multimap} \qquad \frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\ E_{\multimap}$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{U}^n}\ I_0 \qquad \frac{}{\Gamma \vdash s : \mathbb{U}^n \multimap \mathbb{U}^n}\ I_s$$

$$\frac{\Gamma \vdash M : \mathbb{U}^m \multimap A \quad \Delta \vdash N : A \quad \Theta \vdash L : \mathbb{U}^m}{\Gamma, \Delta, \Theta, \Lambda \vdash L\,\{M, N\} : A}\ E^C_{\multimap}$$

$$\frac{\Gamma \vdash M : \mathbb{U}^m \multimap A \multimap A \quad \Delta \vdash N : A \quad \Theta \vdash L : \mathbb{U}^m}{\Gamma, \Delta, \Theta, \Lambda \vdash L\,\langle\!\langle M, N \rangle\!\rangle : A}\ E^R_{\multimap}$$

Types

- Types:

$$A ::= \mathbb{U}^n \mid A \multimap A$$

- Type-assignment rules:

$$\frac{}{\Gamma, x : A \vdash x : A}\ A \qquad \frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash M\{z/x, z/y\} : B}\ C$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ I_{\multimap} \qquad \frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\ E_{\multimap}$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{U}^n}\ I_0 \qquad \frac{}{\Gamma \vdash s : \mathbb{U}^n \multimap \mathbb{U}^n}\ I_s$$

$$\frac{\Gamma \vdash M : \mathbb{U}^m \multimap A \quad \Delta \vdash N : A \quad \Theta \vdash L : \mathbb{U}^m}{\Gamma, \Delta, \Theta, \Lambda \vdash L\,\{M, N\} : A}\ E^C_{\multimap}$$

$$\frac{\Gamma \vdash M : \mathbb{U}^m \multimap A \multimap A \quad \Delta \vdash N : A \quad \Theta \vdash L : \mathbb{U}^m}{\Gamma, \Delta, \Theta, \Lambda \vdash L\,\langle\!\langle M, N \rangle\!\rangle : A}\ E^R_{\multimap}$$

Accomodating Arbitrary Algebras

- The set of base types can be extended to arbitrary free algebras. Examples are
  - **Binary Strings**: Constructors are $\varepsilon, 0, 1$ and have arity $0, 1, 1$, respectively.
  - **Binary Trees**: Constructors are $leaf, node$ and have arity $0, 2$, respectively.
  - **Binary Trees with binary labels**: Constructors are $leaf, nodezero, nodeone$ and have arity $0, 2, 2$, respectively.
- The first one is a word algebra. The other ones are tree algebras.
- $\mathbf{W}$ is the class of word algebras, $\mathbf{A}$ is the class of tree algebras, $\mathbf{T}$ is the class of all types.

Accomodating Arbitrary Algebras

- The set of base types can be extended to arbitrary free algebras. Examples are
  - **Binary Strings**: Constructors are $\varepsilon, 0, 1$ and have arity $0, 1, 1$, respectively.
  - **Binary Trees**: Constructors are $leaf, node$ and have arity $0, 2$, respectively.
  - **Binary Trees with binary labels**: Constructors are $leaf, nodezero, nodeone$ and have arity $0, 2, 2$, respectively.
- The first one is a word algebra. The other ones are tree algebras.
- $\mathbf{W}$ is the class of word algebras, $\mathbf{A}$ is the class of tree algebras, $\mathbf{T}$ is the class of all types.

Accomodating Arbitrary Algebras

- The set of base types can be extended to arbitrary free algebras. Examples are
  - **Binary Strings**: Constructors are $\varepsilon, 0, 1$ and have arity $0, 1, 1$, respectively.
  - **Binary Trees**: Constructors are $leaf, node$ and have arity $0, 2$, respectively.
  - **Binary Trees with binary labels**: Constructors are $leaf, nodezero, nodeone$ and have arity $0, 2, 2$, respectively.
- The first one is a word algebra. The other ones are tree algebras.
- $\mathbf{W}$ is the class of word algebras, $\mathbf{A}$ is the class of tree algebras, $\mathbf{T}$ is the class of all types.

Accomodating Arbitrary Algebras

- The set of base types can be extended to arbitrary free algebras. Examples are
  - **Binary Strings**: Constructors are $\varepsilon, 0, 1$ and have arity $0, 1, 1$, respectively.
  - **Binary Trees**: Constructors are $leaf, node$ and have arity $0, 2$, respectively.
  - **Binary Trees with binary labels**: Constructors are $leaf, nodezero, nodeone$ and have arity $0, 2, 2$, respectively.
- The first one is a word algebra. The other ones are tree algebras.
- $\mathbf{W}$ is the class of word algebras, $\mathbf{A}$ is the class of tree algebras, $\mathbf{T}$ is the class of all types.

Accomodating Arbitrary Algebras

- The set of base types can be extended to arbitrary free algebras. Examples are
  - **Binary Strings**: Constructors are $\varepsilon, 0, 1$ and have arity $0, 1, 1$, respectively.
  - **Binary Trees**: Constructors are $leaf, node$ and have arity $0, 2$, respectively.
  - **Binary Trees with binary labels**: Constructors are $leaf, nodezero, nodeone$ and have arity $0, 2, 2$, respectively.
- The first one is a word algebra. The other ones are tree algebras.
- $\mathbf{W}$ is the class of word algebras, $\mathbf{A}$ is the class of tree algebras, $\mathbf{T}$ is the class of all types.

Subsystems

- Expressive power of the whole calculus: too large!.
- We are interested in two different constraints on the type-system:
  - We can allow **contraction** to be applicable only to some types (in the style of linear logic [Girard87]).
  - We can introduce **ramification** (similarly to [Hofmann97])
- In this way, we can define various subsystems. For example:
  - A subsystem with ramification where contraction is applicable only to word algebras: $\mathbf{RH}(\mathbf{W})$.
  - A subsystem without ramification where contraction cannot be applied: $\mathbf{H}(\emptyset)$.

  All these fragments enjoy subject-reduction.

Subsystems

- Expressive power of the whole calculus: too large!.
- We are interested in two different constraints on the type-system:
  - We can allow **contraction** to be applicable only to some types (in the style of linear logic [Girard87]).
  - We can introduce **ramification** (similarly to [Hofmann97])
- In this way, we can define various subsystems. For example:
  - A subsystem with ramification where contraction is applicable only to word algebras: $\mathbf{RH}(\mathbf{W})$.
  - A subsystem without ramification where contraction cannot be applied: $\mathbf{H}(\emptyset)$.

  All these fragments enjoy subject-reduction.

Subsystems

- Expressive power of the whole calculus: too large!.
- We are interested in two different constraints on the type-system:
  - We can allow **contraction** to be applicable only to some types (in the style of linear logic [Girard87]).
  - We can introduce **ramification** (similarly to [Hofmann97])
- In this way, we can define various subsystems. For example:
  - A subsystem with ramification where contraction is applicable only to word algebras: $\mathbf{RH}(\mathbf{W})$.
  - A subsystem without ramification where contraction cannot be applied: $\mathbf{H}(\emptyset)$.

  All these fragments enjoy subject-reduction.

The Results

- Here are the obtained results:

	$\mathbf{T}$	$\mathbf{A}$	$\mathbf{W}$	$\emptyset$
$\mathbf{H}(\cdot)$	PA	PR	PR	PR
$\mathbf{RH}(\cdot)$	E	E	P	P

- Going from $\mathbf{RH}(\mathbf{W})$ to $\mathbf{RH}(\mathbf{A})$ greatly increases the expressive power.
- Going from $\mathbf{RH}(\emptyset)$ to $\mathbf{RH}(\mathbf{W})$ does not change anything.
- Forbidding contraction on higher-order types (almost always) "neutralizes" higher-order recursion.

Parameters Controlling Computing Time - I

- All the results we have cited are characterization results: both soundness and completeness hold.
- Two parameters control computing time:
  - The **recursion depth** $R(M)$ of a term M is the nesting depth of recursions inside M.
  - The **highest tier** $I(\pi)$ of a type derivation π is the maximum integer i such that there is an instance of $E^R_{\multimap}$ in π with the shape

$$\frac{\pi_1 \quad \dots \quad \pi_n \quad \Delta \vdash L : \mathbb{A}^i}{\Gamma, \Delta \vdash L \langle\langle M_1, \dots, M_n \rangle\rangle : C}$$

Parameters Controlling Computing Time - I

- All the results we have cited are characterization results: both soundness and completeness hold.
- Two parameters control computing time:
  - The **recursion depth** $R(M)$ of a term M is the nesting depth of recursions inside M.
  - The **highest tier** $I(\pi)$ of a type derivation π is the maximum integer i such that there is an instance of $E^R_{\multimap}$ in π with the shape

$$\frac{\pi_1 \quad \ldots \quad \pi_n \quad \Delta \vdash L : \mathbb{A}^i}{\Gamma, \Delta \vdash L\,\langle\!\langle M_1, \ldots, M_n \rangle\!\rangle : C}$$

Parameters Controlling Computing Time - II

- From the following, you can easily understand in which sense the recursion depth and the highest tier control the computing time:

Theorem (Polytime soundness for $\mathbf{RH(W)}$)

For every $n, m \in \mathbb{N}$, there are polynomials p_m^n such that for every type derivation $\pi : \Gamma \vdash_{\mathbf{RH(W)}} M : A$, the time needed to normalize M is at most then $p_{R(M)}^{I(\pi)}(|M|)$.

- For any other fragment, the soundness theorem is almost identical. What changes is the class which p_m^n belongs to.

Parameters Controlling Computing Time - II

- From the following, you can easily understand in which sense the recursion depth and the highest tier control the computing time:

Theorem (Polytime soundness for $\mathbf{RH(W)}$)

For every $n, m \in \mathbb{N}$, there are polynomials p_m^n such that for every type derivation $\pi : \Gamma \vdash_{\mathbf{RH(W)}} M : A$, the time needed to normalize M is at most then $p_{R(M)}^{I(\pi)}(|M|)$.

- For any other fragment, the soundness theorem is almost identical. What changes is the class which p_m^n belongs to.

A Crucial Result

- The algebraic potential size $A(M)$ of a term M is is the maximum natural number n such that $M \rightsquigarrow^* N$ and there is a redex in N whose argument is a free algebra term t with $|t| = n$.
- We get:

Theorem

For every $d \in \mathbb{N}$ there are polynomials $p_d, q_d : \mathbb{N}^2 \to \mathbb{N}$ such that whenever $\pi : \Gamma \vdash M : A$ and $M \rightsquigarrow^n N$, then $n \leq p_{R(M)}(|M|, A(M))$ and $|N| \leq q_{R(M)}(|M|, A(M))$

- Knowing the algebraic potential size of terms means knowing a lot about the complexity of normalization!

A Crucial Result

- The algebraic potential size $A(M)$ of a term M is is the maximum natural number n such that $M \rightsquigarrow^* N$ and there is a redex in N whose argument is a free algebra term t with $|t| = n$.
- We get:

Theorem

For every $d \in \mathbb{N}$ there are polynomials $p_d, q_d : \mathbb{N}^2 \to \mathbb{N}$ such that whenever $\pi : \Gamma \vdash M : A$ and $M \rightsquigarrow^n N$, then $n \leq p_{R(M)}(|M|, A(M))$ and $|N| \leq q_{R(M)}(|M|, A(M))$

- Knowing the algebraic potential size of terms means knowing a lot about the complexity of normalization!

A Crucial Result

- The algebraic potential size $A(M)$ of a term M is is the maximum natural number n such that $M \rightsquigarrow^* N$ and there is a redex in N whose argument is a free algebra term t with $|t| = n$.
- We get:

Theorem

For every $d \in \mathbb{N}$ there are polynomials $p_d, q_d : \mathbb{N}^2 \to \mathbb{N}$ such that whenever $\pi : \Gamma \vdash M : A$ and $M \rightsquigarrow^n N$, then $n \leq p_{R(M)}(|M|, A(M))$ and $|N| \leq q_{R(M)}(|M|, A(M))$

- Knowing the algebraic potential size of terms means knowing a lot about the complexity of normalization!

Context Semantics in Five Minutes

- Type derivations as graphs. From a type derivation π we can obtaine a graph G_π.
- For every graph, a set of trees. From a graph G, a set of trees $\mathcal{T}(G)$.
- There is a morphism Φ extracting a free algebra term t from every tree in $\mathcal{T}(G)$.
- If $\pi : \Gamma \vdash M : A$, $M \rightsquigarrow^* N$ and t appear as an argument of a redex in N, then $t \in \Phi(\mathcal{T}(G_\pi))$.
- Interestingly:
  - The size of G_π is proportional to that of π.
  - The size of $\Phi(T)$ is proportional to that of T.
  - Studying the combinatorial behaviour of $\mathcal{T}(\cdot)$ as a map suffices!

Context Semantics in Five Minutes

- Type derivations as graphs. From a type derivation π we can obtaine a graph G_π.
- For every graph, a set of trees. From a graph G, a set of trees $\mathcal{T}(G)$.
- There is a morphism Φ extracting a free algebra term t from every tree in $\mathcal{T}(G)$.
- If $\pi : \Gamma \vdash M : A$, $M \rightsquigarrow^* N$ and t appear as an argument of a redex in N, then $t \in \Phi(\mathcal{T}(G_\pi))$.
- Interestingly:
  - The size of G_π is proportional to that of π.
  - The size of $\Phi(T)$ is proportional to that of T.
  - Studying the combinatorial behaviour of $\mathcal{T}(\cdot)$ as a map suffices!

Context Semantics in Five Minutes

- Type derivations as graphs. From a type derivation π we can obtaine a graph G_π.
- For every graph, a set of trees. From a graph G, a set of trees $\mathcal{T}(G)$.
- There is a morphism Φ extracting a free algebra term t from every tree in $\mathcal{T}(G)$.
- If $\pi : \Gamma \vdash M : A$, $M \rightsquigarrow^* N$ and t appear as an argument of a redex in N, then $t \in \Phi(\mathcal{T}(G_\pi))$.
- Interestingly:
  - The size of G_π is proportional to that of π.
  - The size of $\Phi(T)$ is proportional to that of T.
  - Studying the combinatorial behaviour of $\mathcal{T}(\cdot)$ as a map suffices!

Context Semantics in Five Minutes

- Type derivations as graphs. From a type derivation π we can obtaine a graph G_π.
- For every graph, a set of trees. From a graph G, a set of trees $\mathcal{T}(G)$.
- There is a morphism Φ extracting a free algebra term t from every tree in $\mathcal{T}(G)$.
- If $\pi : \Gamma \vdash M : A$, $M \rightsquigarrow^* N$ and t appear as an argument of a redex in N, then $t \in \Phi(\mathcal{T}(G_\pi))$.
- Interestingly:
  - The size of G_π is proportional to that of π.
  - The size of $\Phi(T)$ is proportional to that of T.
  - Studying the combinatorial behaviour of $\mathcal{T}(\cdot)$ as a map suffices!

Context Semantics in Five Minutes

- Type derivations as graphs. From a type derivation π we can obtaine a graph G_π.
- For every graph, a set of trees. From a graph G, a set of trees $\mathcal{T}(G)$.
- There is a morphism Φ extracting a free algebra term t from every tree in $\mathcal{T}(G)$.
- If $\pi : \Gamma \vdash M : A$, $M \rightsquigarrow^* N$ and t appear as an argument of a redex in N, then $t \in \Phi(\mathcal{T}(G_\pi))$.
- Interestingly:
  - The size of G_π is proportional to that of π.
  - The size of $\Phi(T)$ is proportional to that of T.
  - Studying the combinatorial behaviour of $\mathcal{T}(\cdot)$ as a map suffices!

Trees in $\mathcal{T}(G)$

- Branches correspond to execution paths.
- Nodes are labelled with contexts

$$(A[\cdot], (u_1[\cdot], t_1) \ldots (u_n[\cdot], t_n))$$

  where:
  - $A[\cdot]$ is a type-context guiding tree costruction;
  - each $(u_i[\cdot], t_i)$ keeps track of which step function copy we are visiting. $u_i[\cdot]$ is a free-algebra-term-context, while t_i is a free algebra term.
- This is somehow reminiscent of token machines and game semantics.

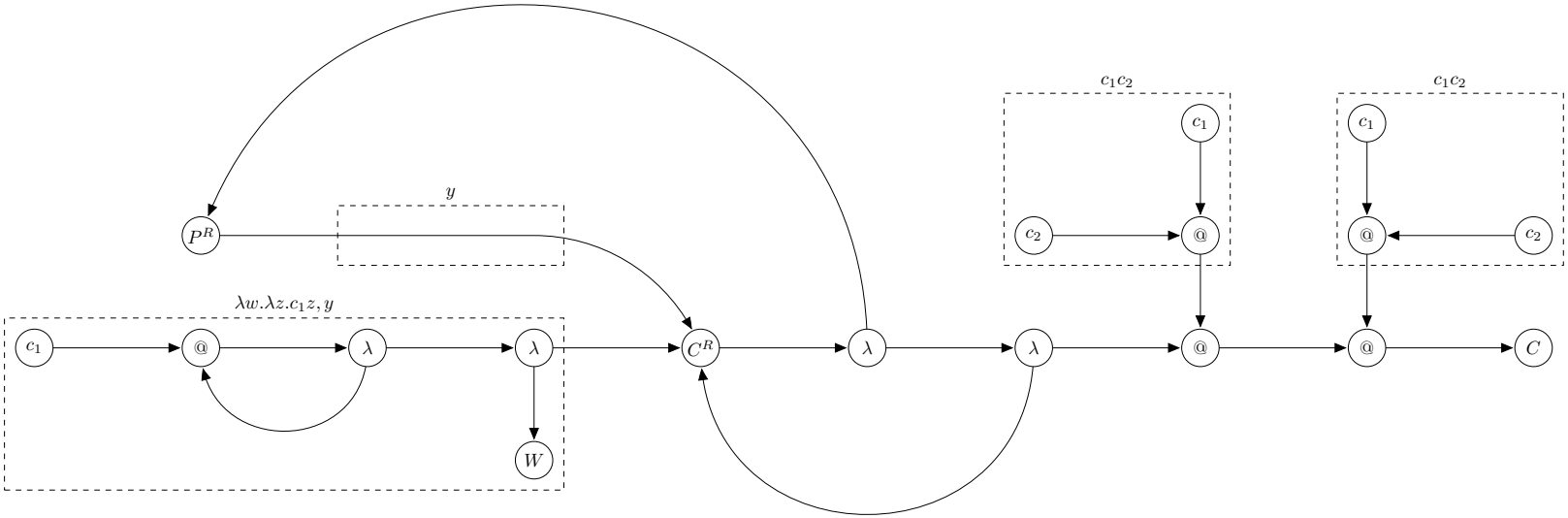
An Example

- Consider $\mathbf{UnAdd} \equiv \lambda x.\lambda y.x\langle\langle \lambda w.\lambda z.c_1^{\mathbb{U}} z, y\rangle\rangle$.
  $\mathbf{UnAdd}\lceil n\rceil\lceil m\rceil \rightsquigarrow^* \lceil n+m\rceil$ for every $n, m \in \mathbb{N}$.
- Consider the following sketch of a type derivation for $\mathbf{UnAdd}\lceil 1\rceil\lceil 1\rceil$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w:\mathbb{U}^1 \vdash c_1^{\mathbb{U}}} \quad \dfrac{}{z:\mathbb{U}^0 \vdash z:\mathbb{U}^0}}{w:\mathbb{U}^1, z:\mathbb{U}^0 \vdash c_1^{\mathbb{U}} z:\mathbb{U}^0}}{w:\mathbb{U}^1 \vdash \lambda z.c_1^{\mathbb{U}} z:\mathbb{U}^0 \multimap \mathbb{U}^0}}{\vdash \lambda w.\lambda z.c_1^{\mathbb{U}} z:\mathbb{U}^1 \multimap \mathbb{U}^0 \multimap \mathbb{U}^0} \quad \dfrac{}{y:\mathbb{U}^0 \vdash y:\mathbb{U}^0} \quad \dfrac{}{x_{\mathbb{U}}^1 \vdash x:\mathbb{U}^1}}{x:\mathbb{U}^1, y:\mathbb{U}^0 \vdash x\langle\langle \lambda w.\lambda z.c_1^{\mathbb{U}} z, y\rangle\rangle:\mathbb{U}^0}}{\vdash \mathbf{UnAdd}:\mathbb{U}^1 \multimap \mathbb{U}^0 \multimap \mathbb{U}^0} \qquad \dfrac{}{\vdash \lceil 1\rceil:\mathbb{U}^1} \quad \dfrac{}{\vdash \lceil 1\rceil:\mathbb{U}^0}}{\vdash \mathbf{UnAdd}\lceil 1\rceil\lceil 1\rceil:\mathbb{U}^0}$$

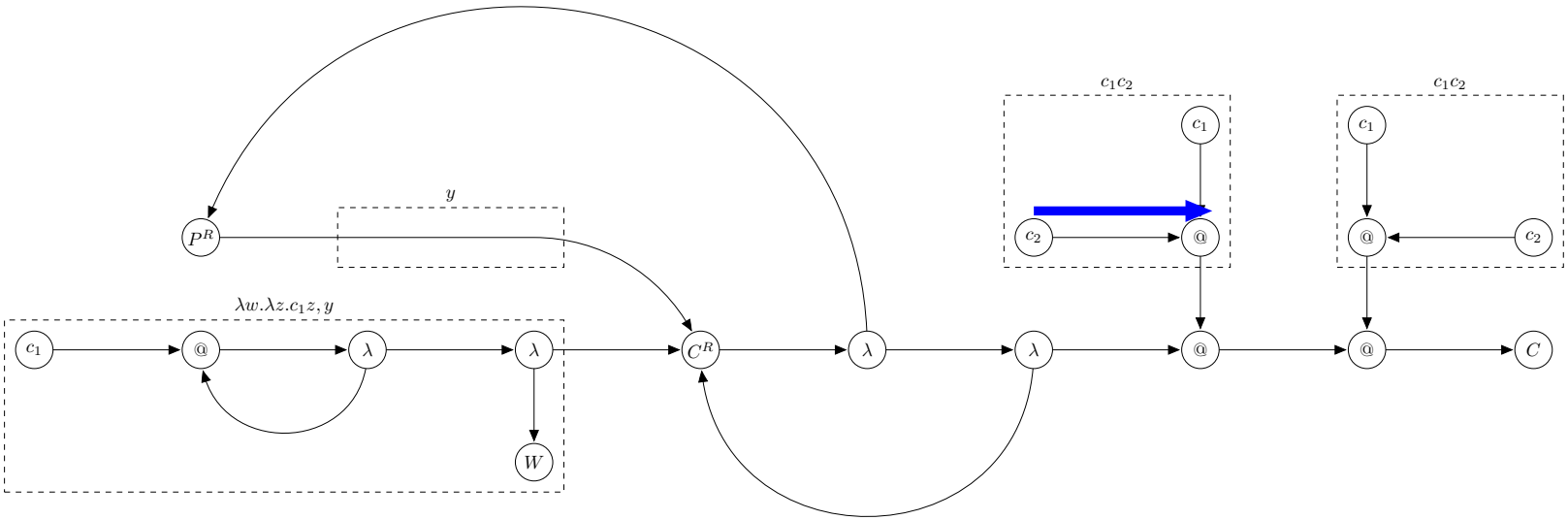
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



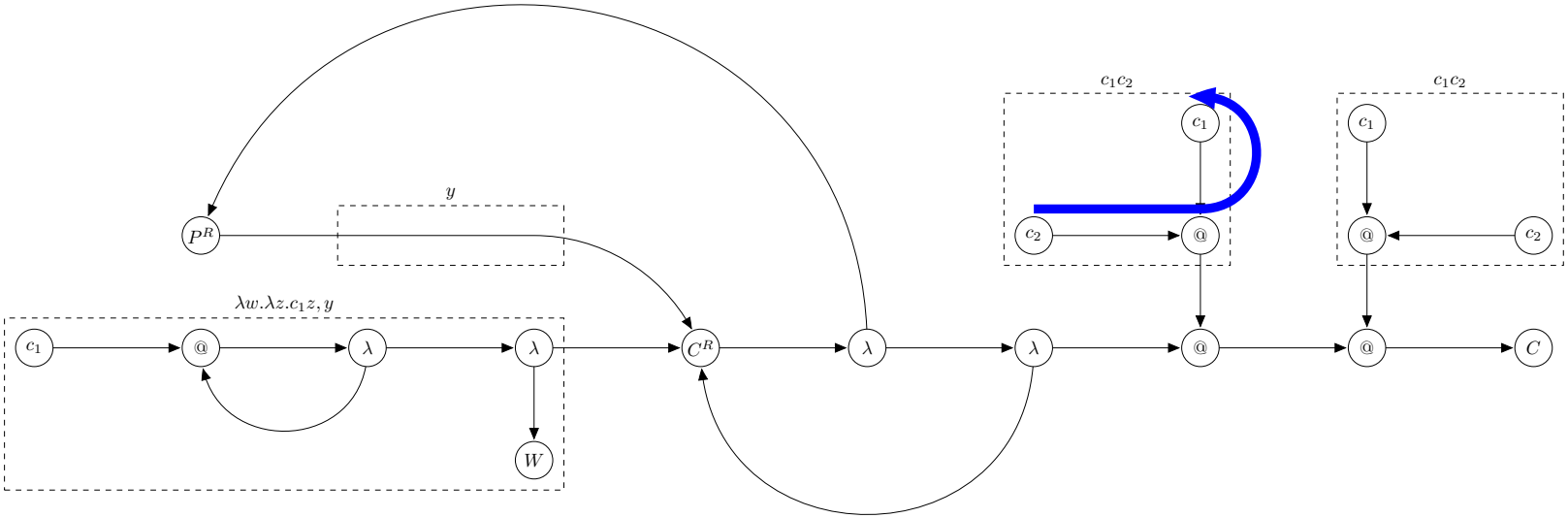
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



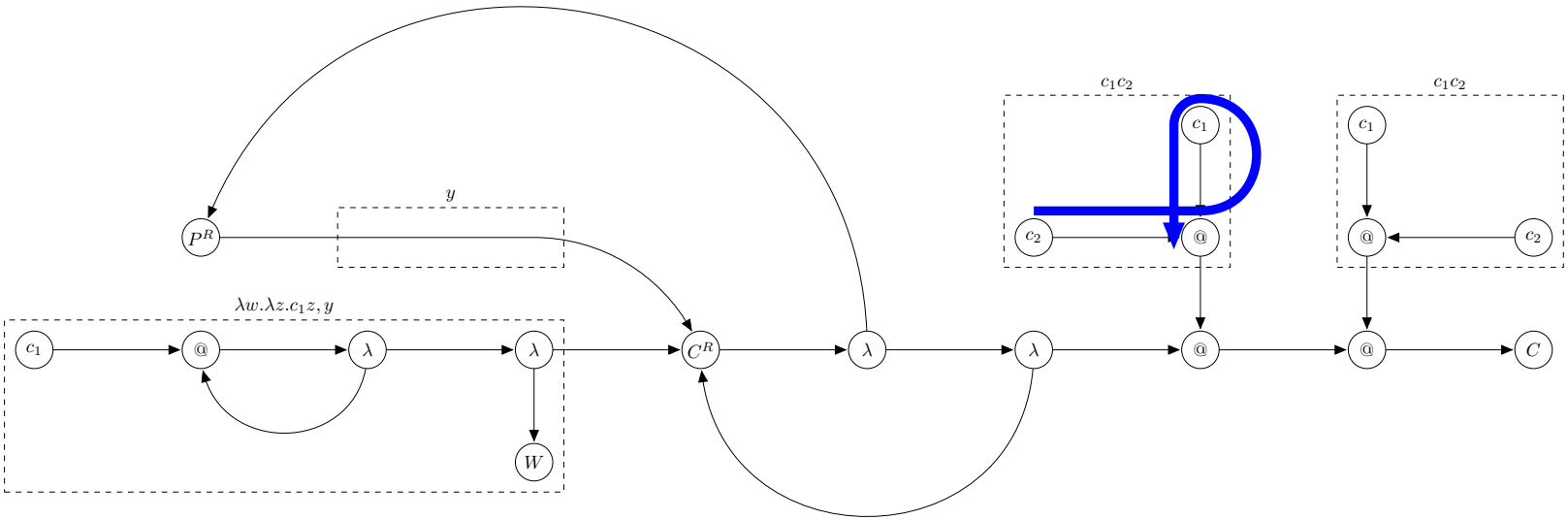
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



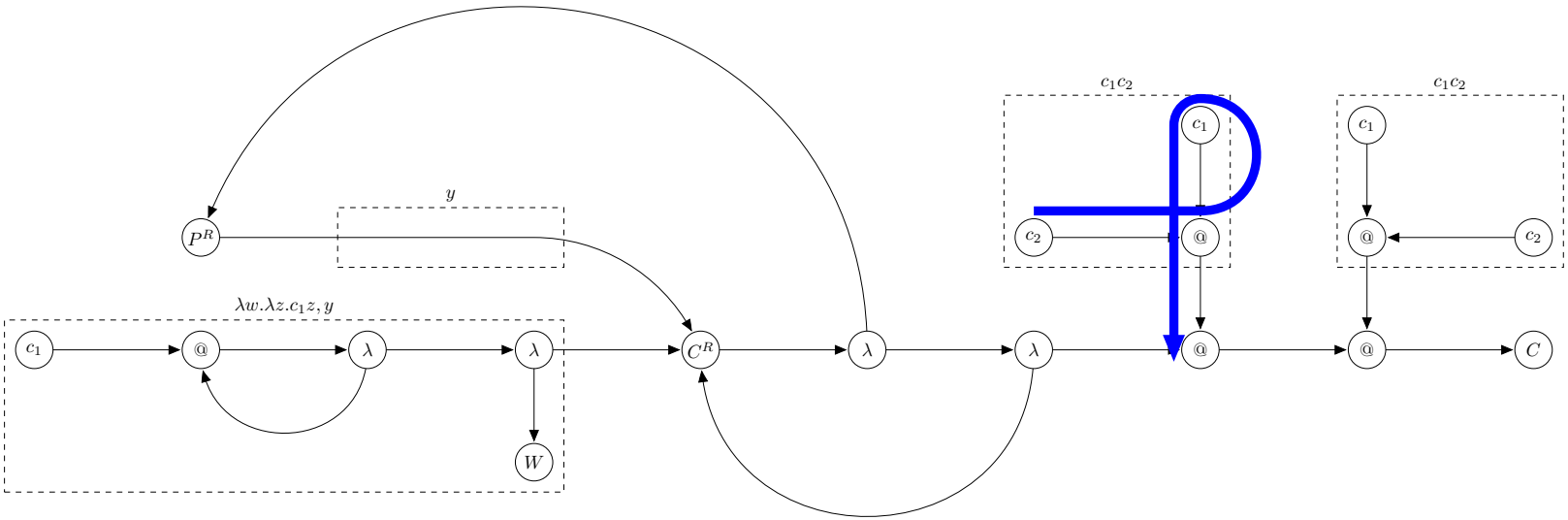
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



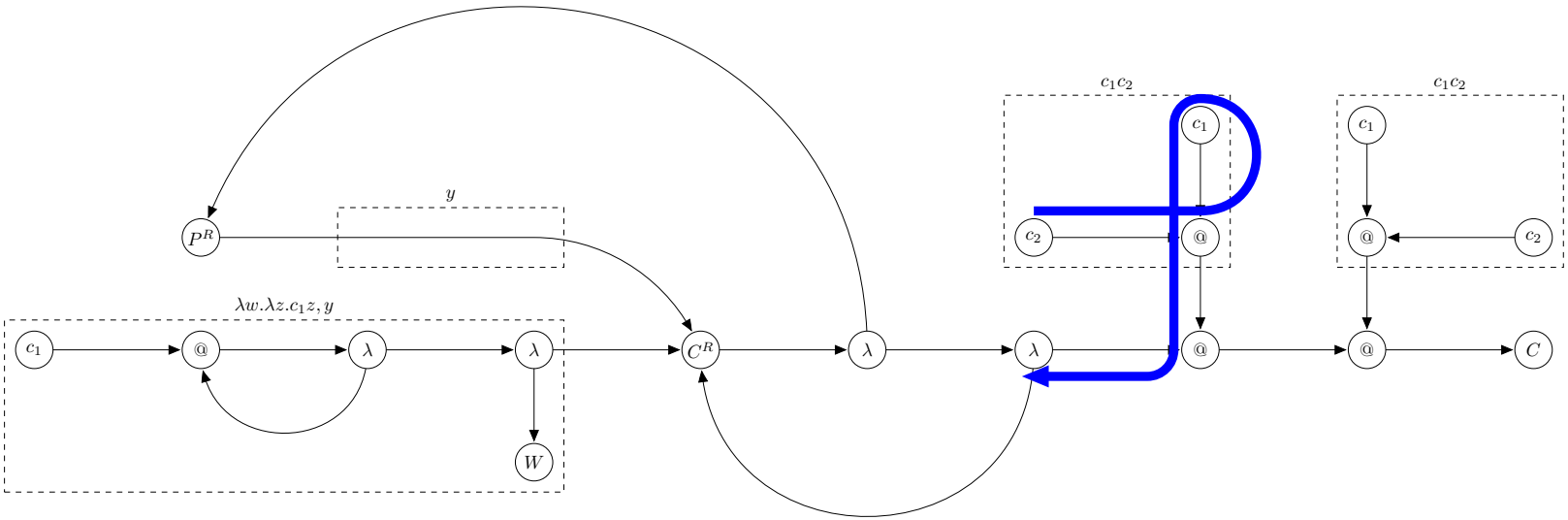
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



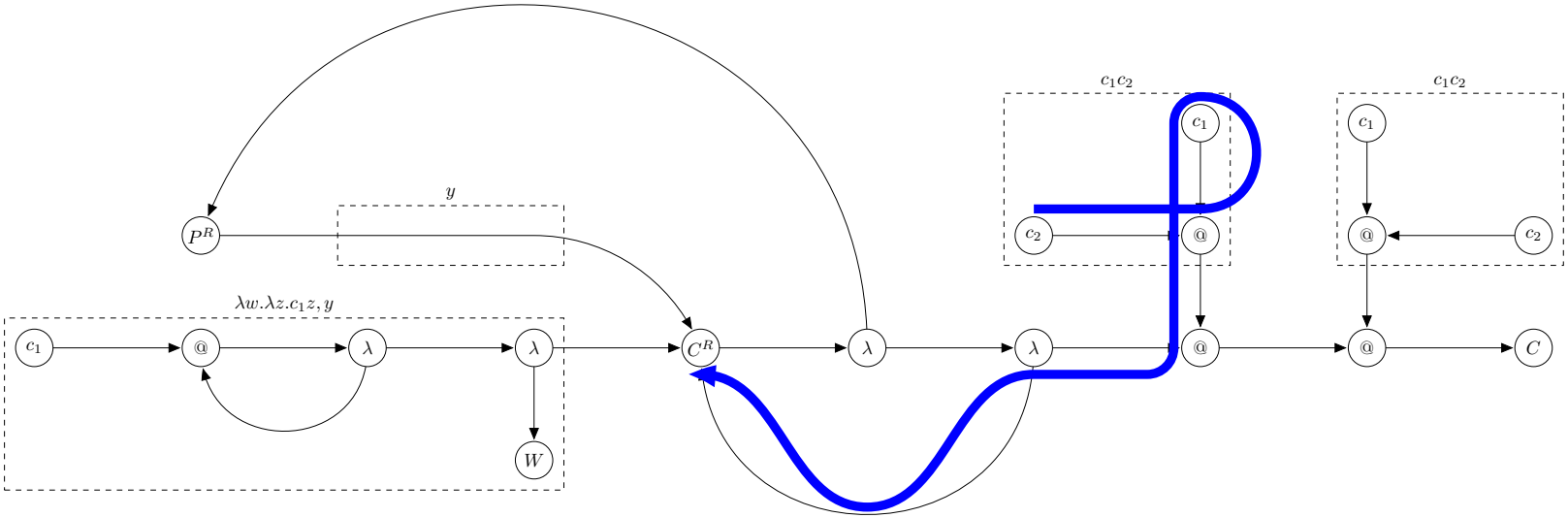
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



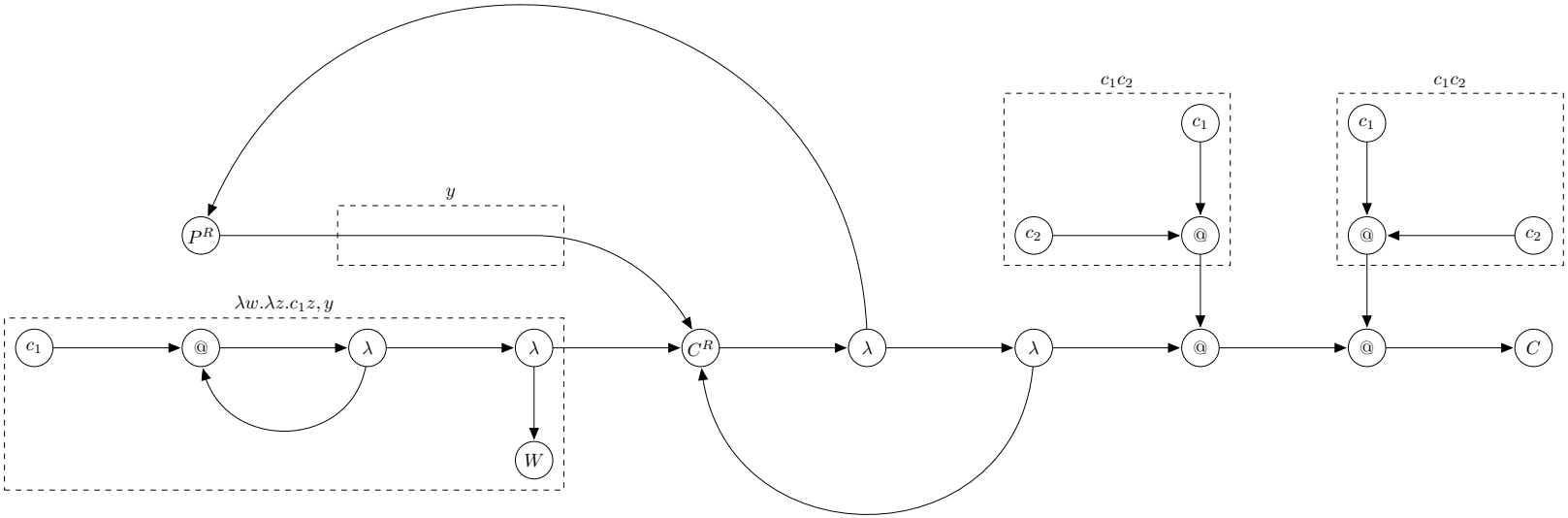
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



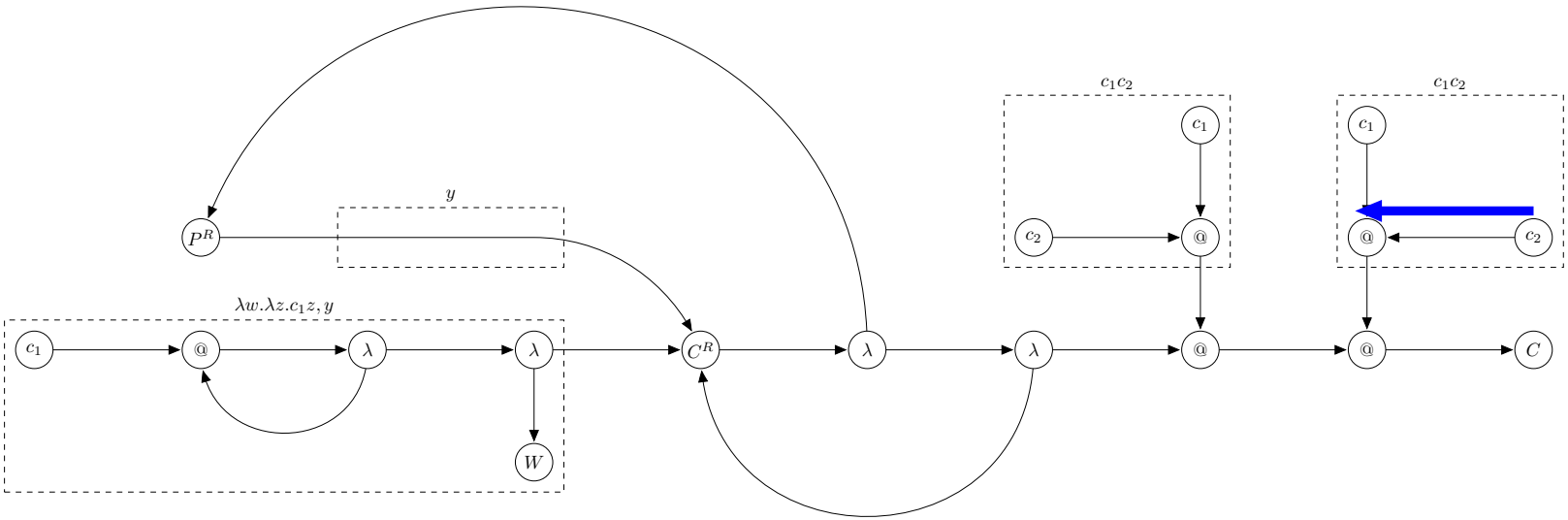
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



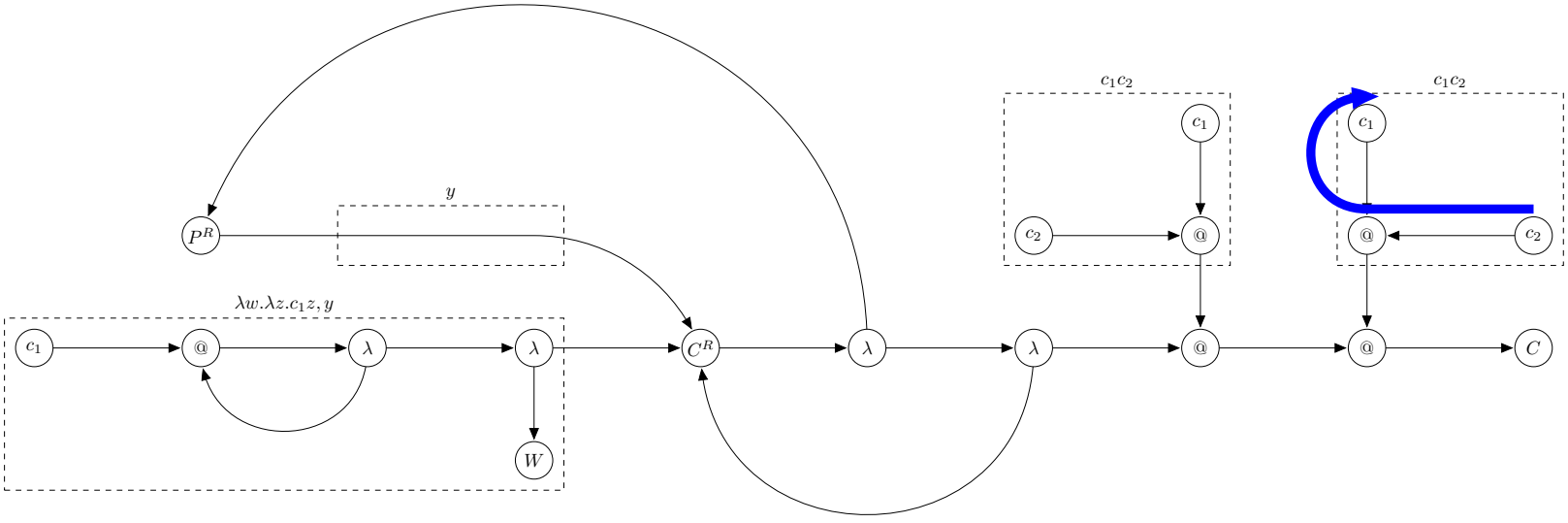
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



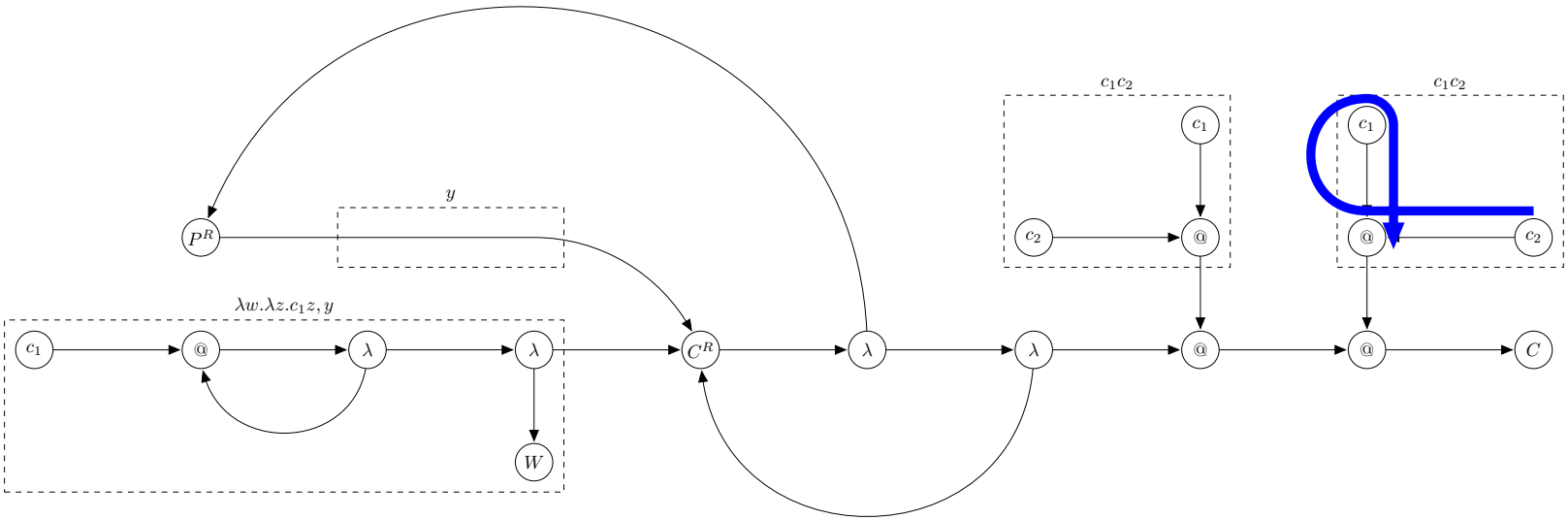
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



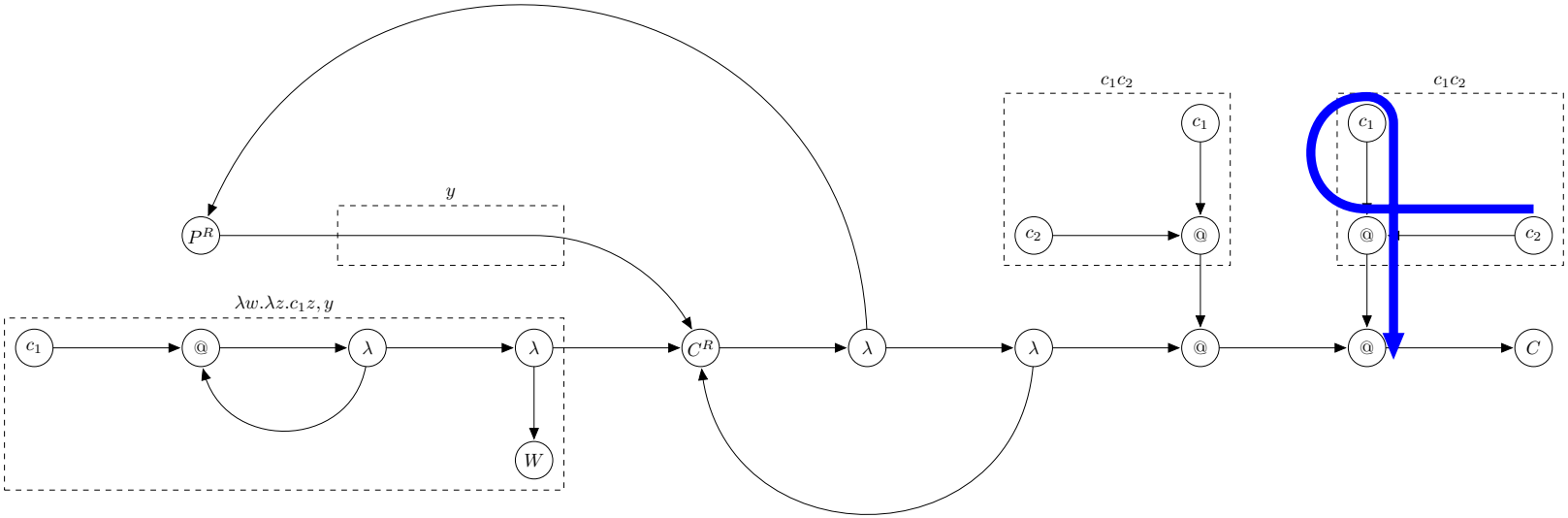
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



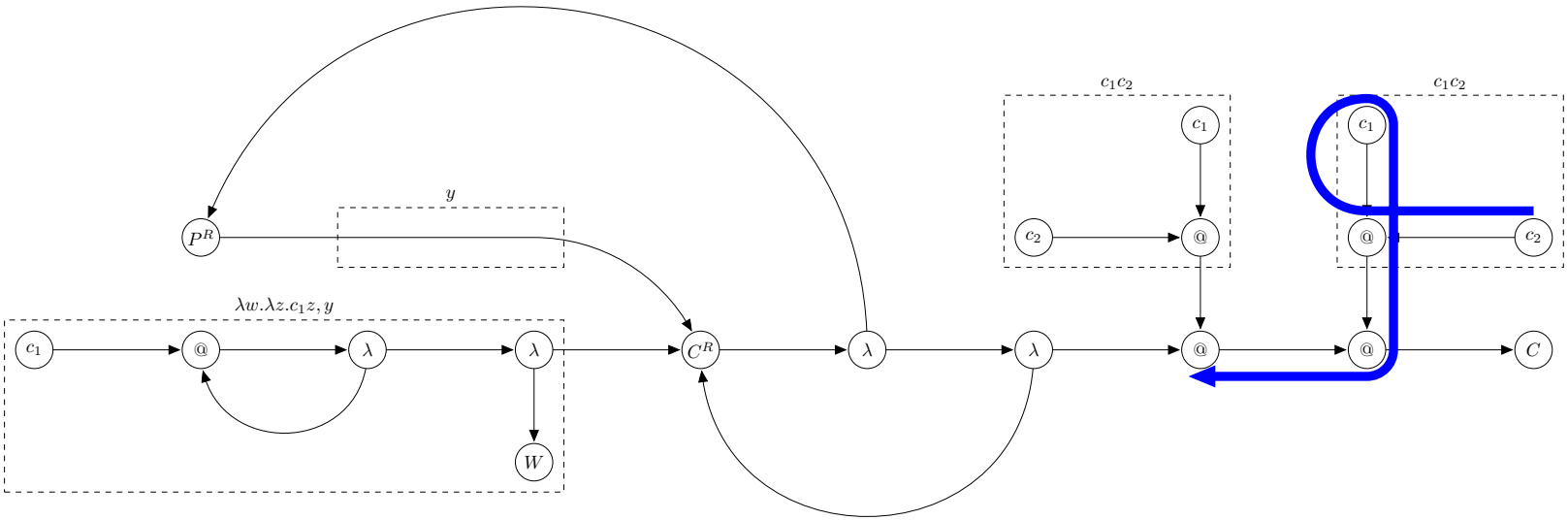
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



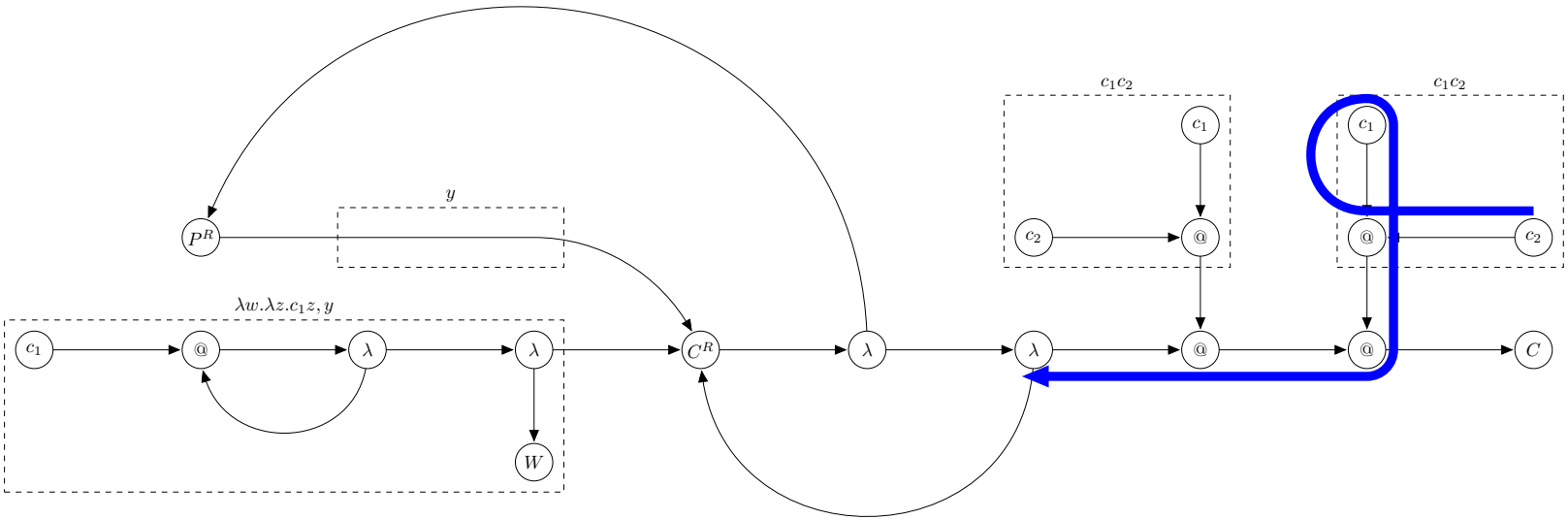
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



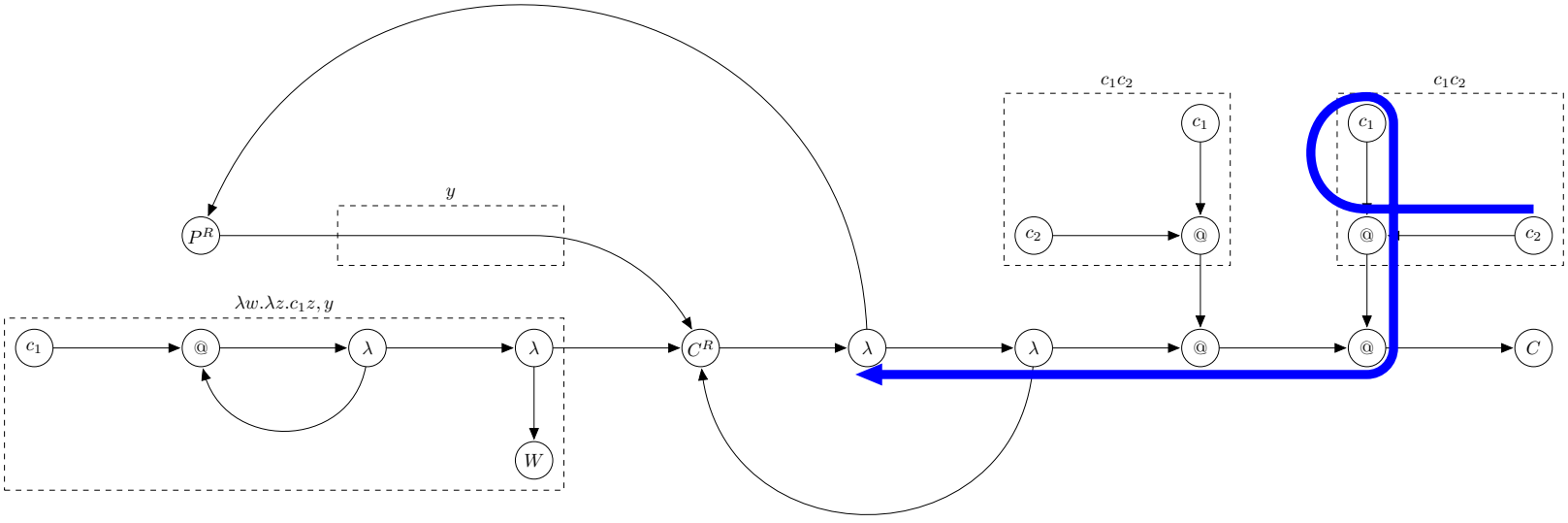
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



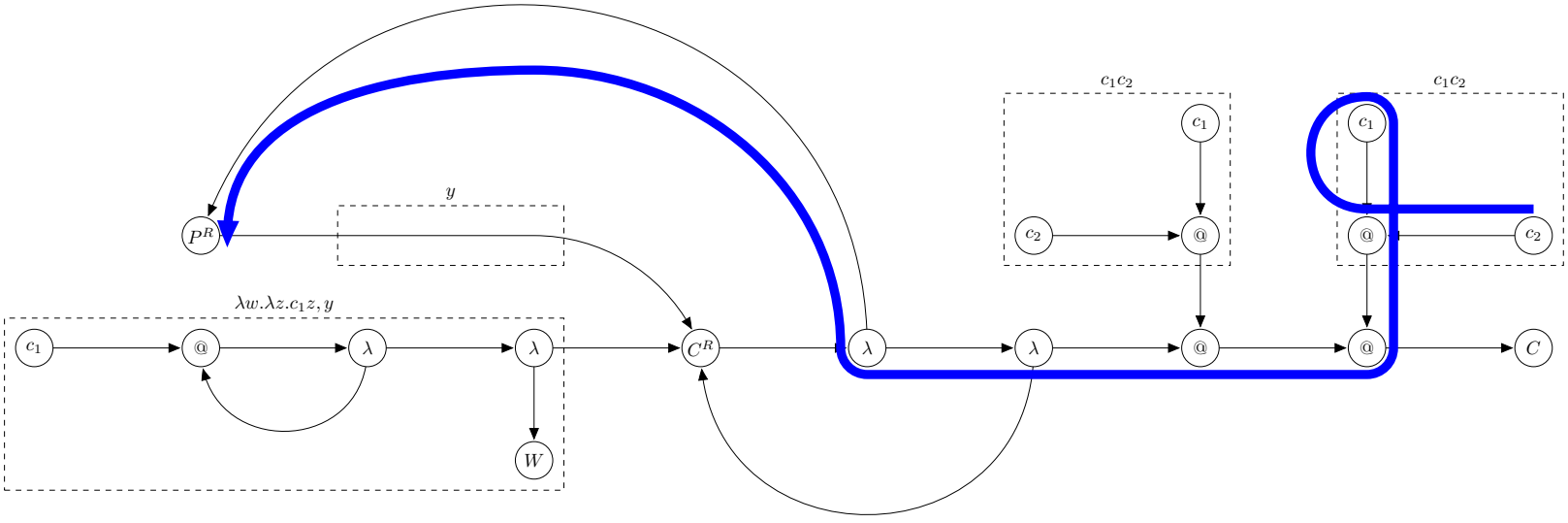
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



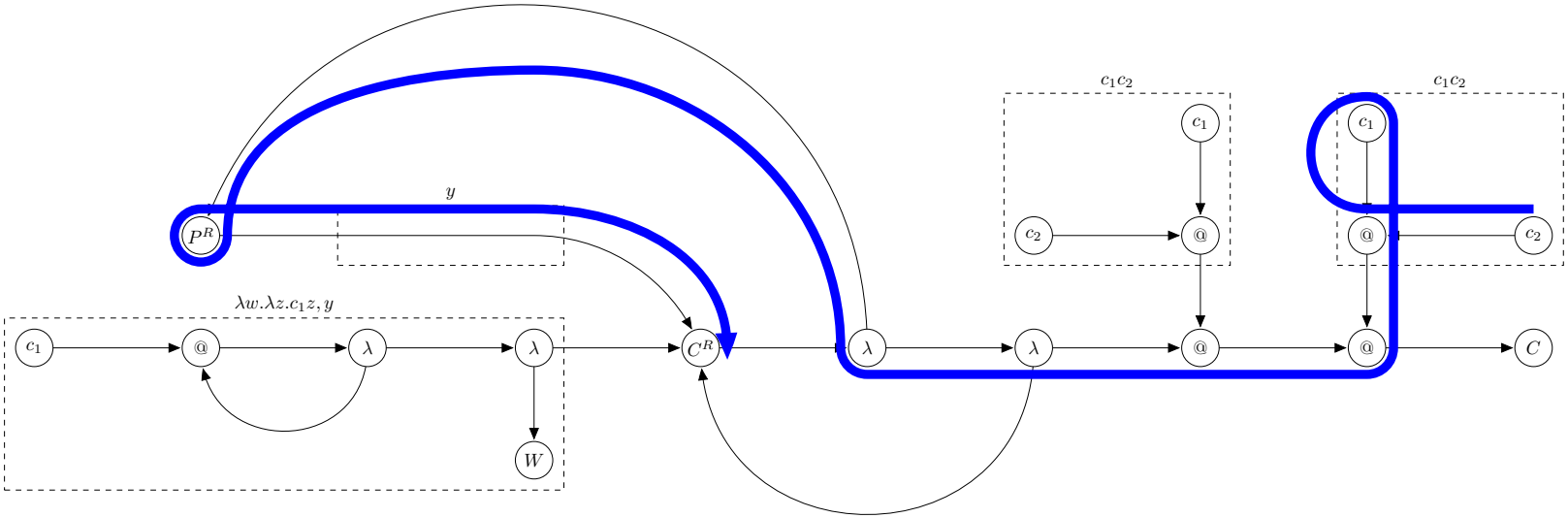
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



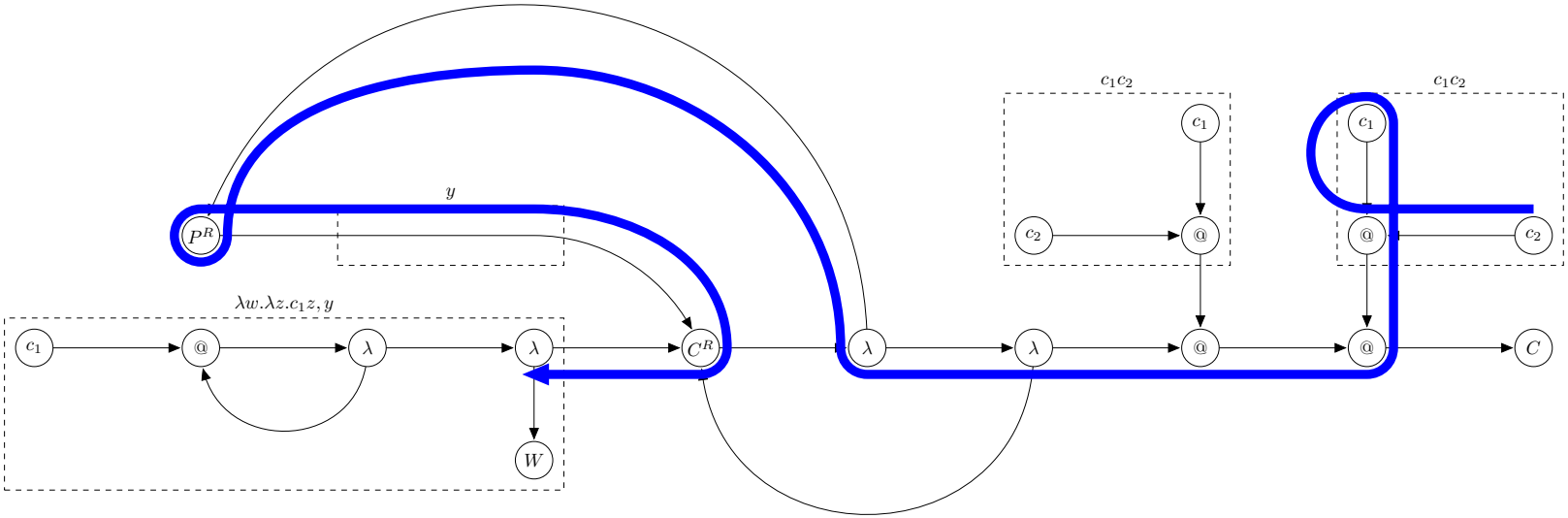
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



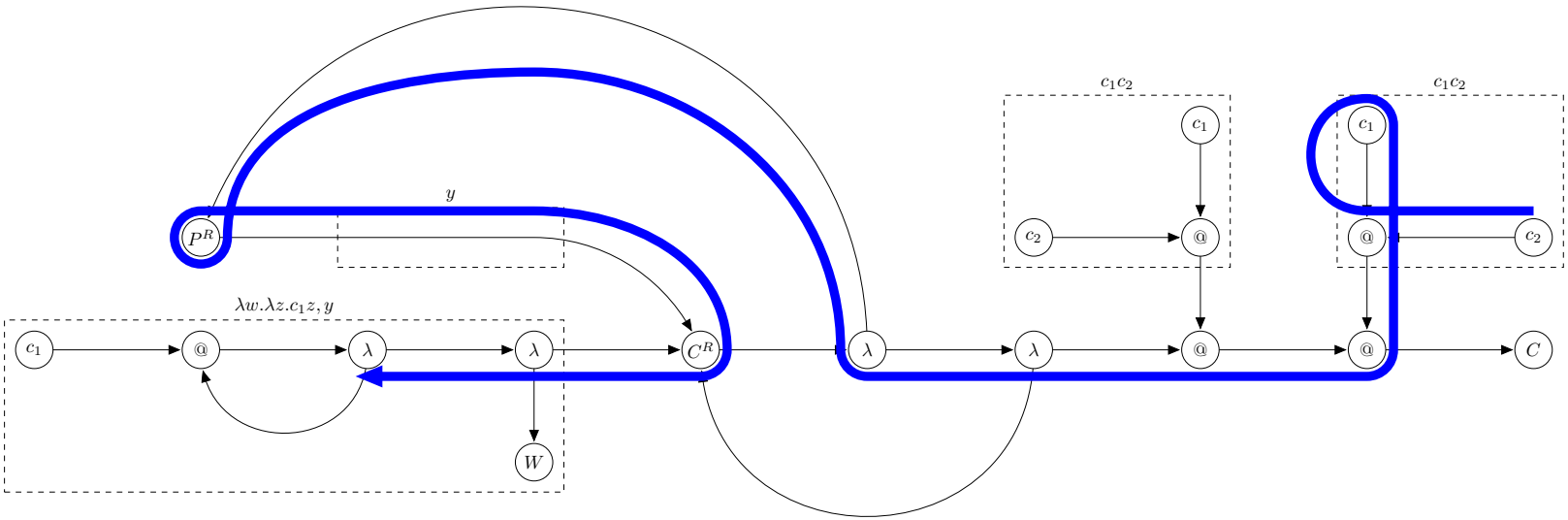
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



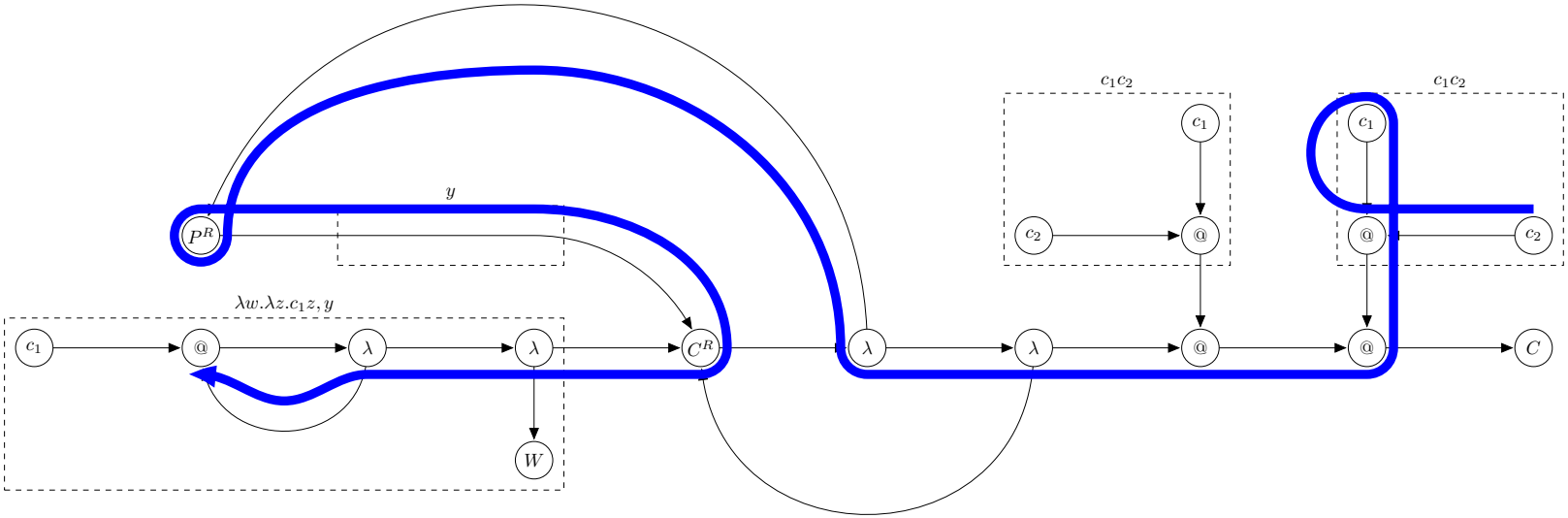
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



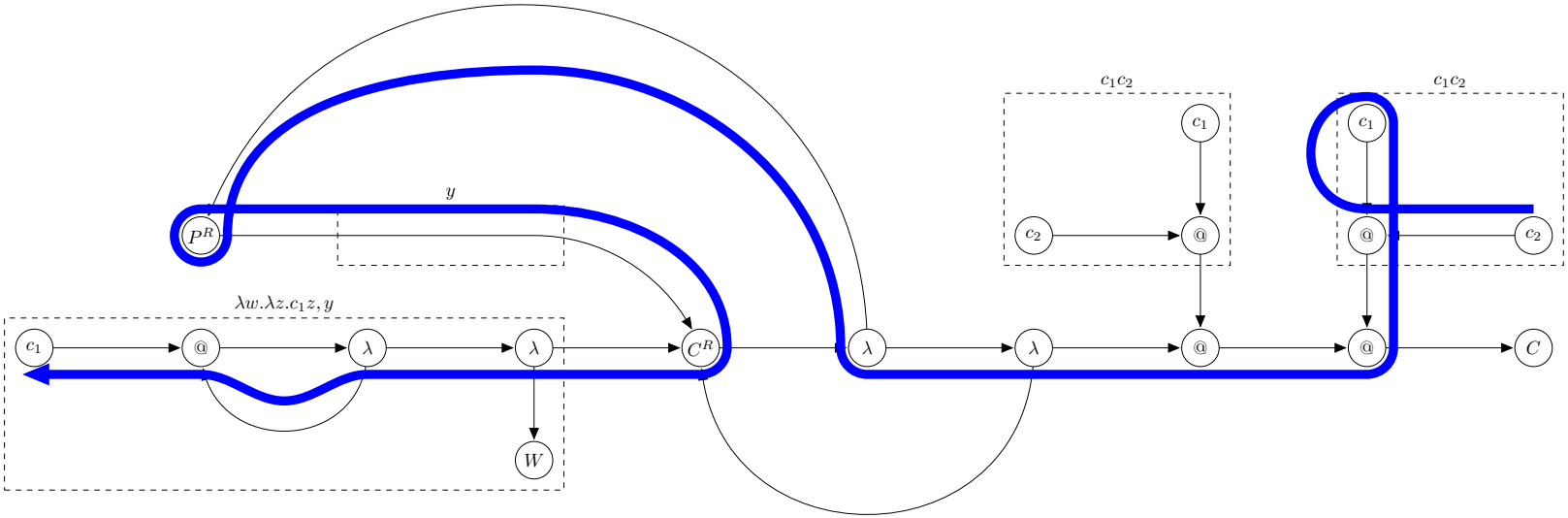
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



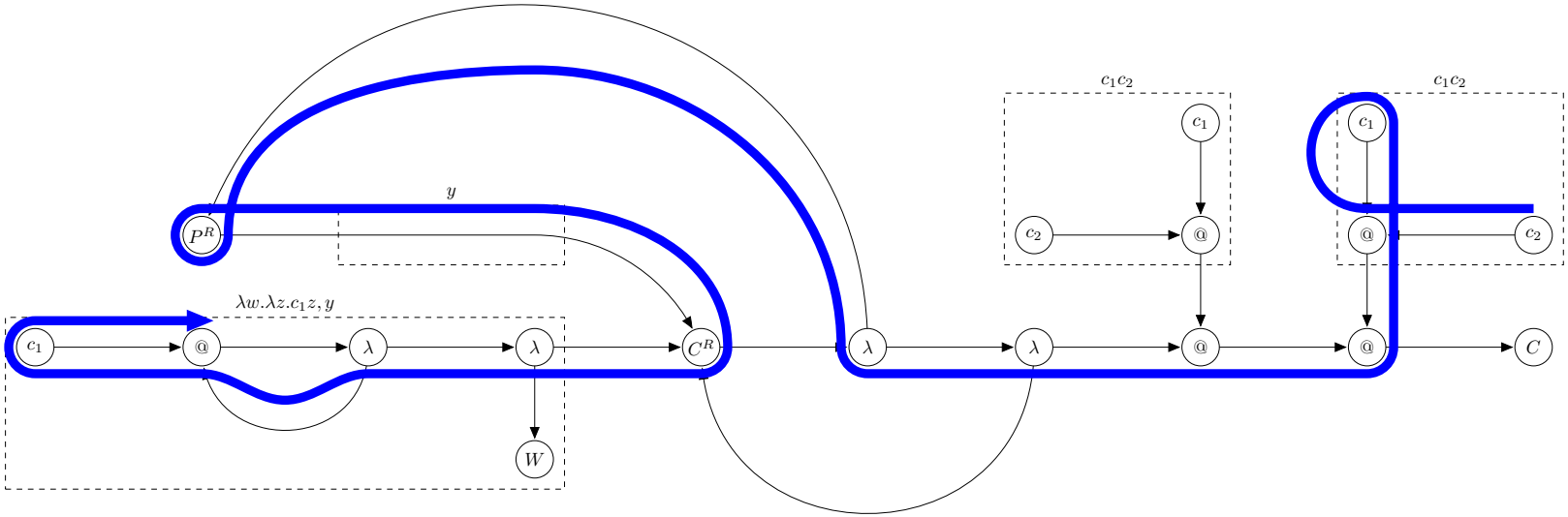
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



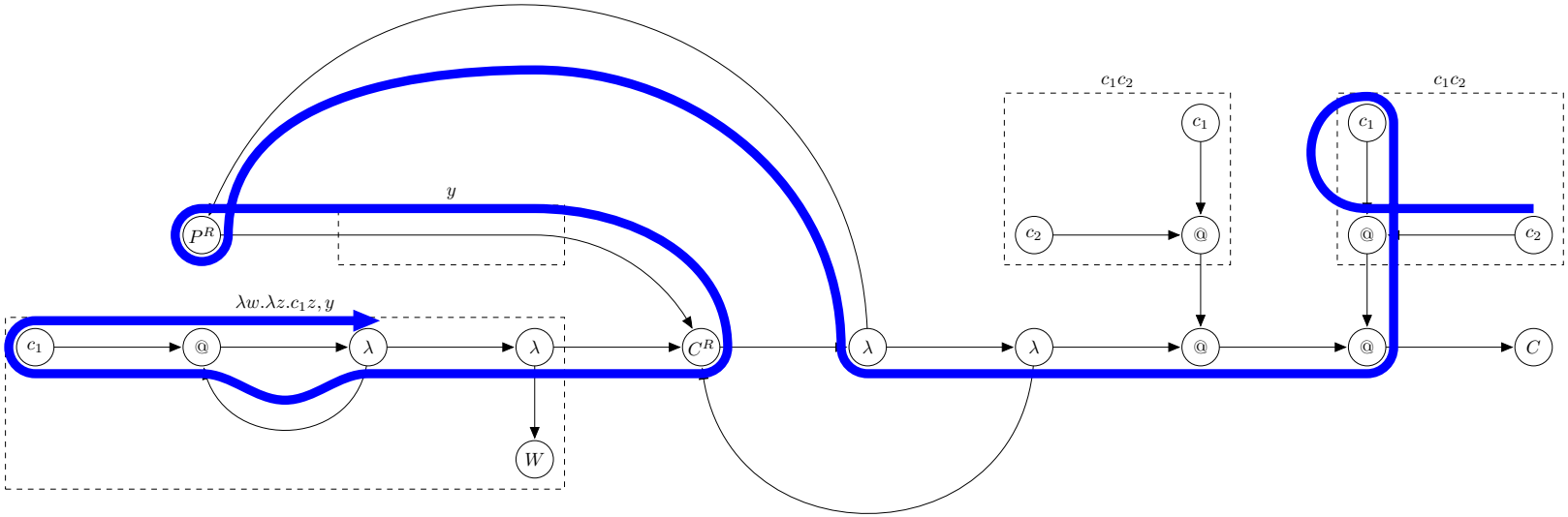
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



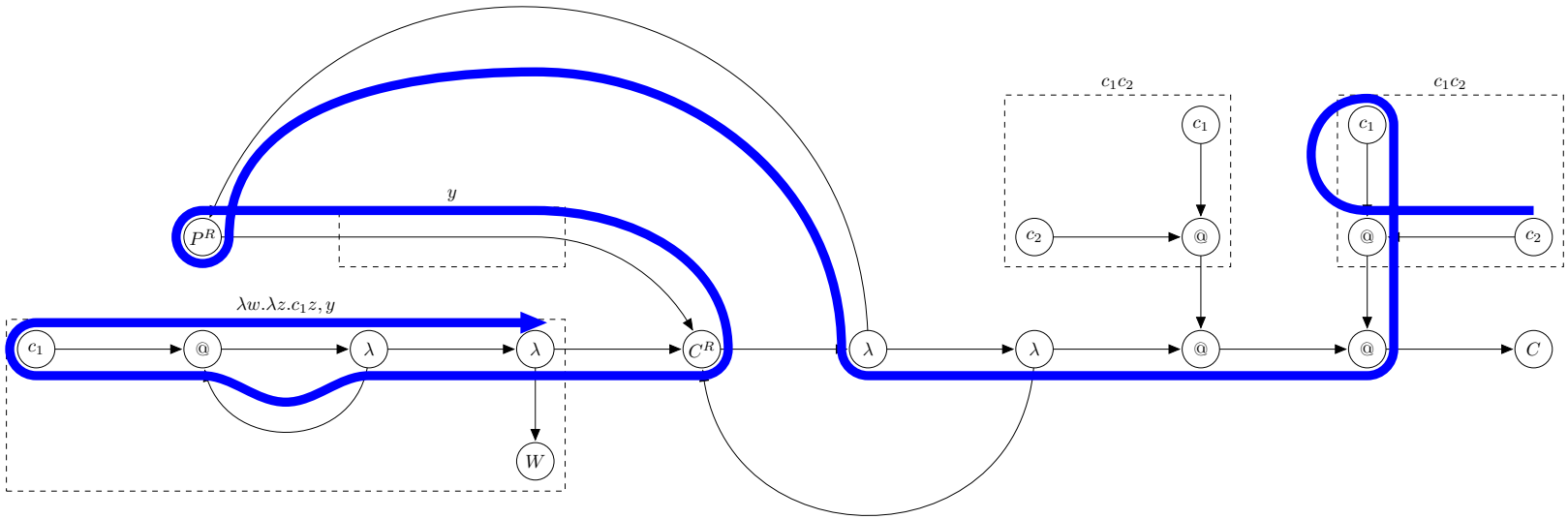
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



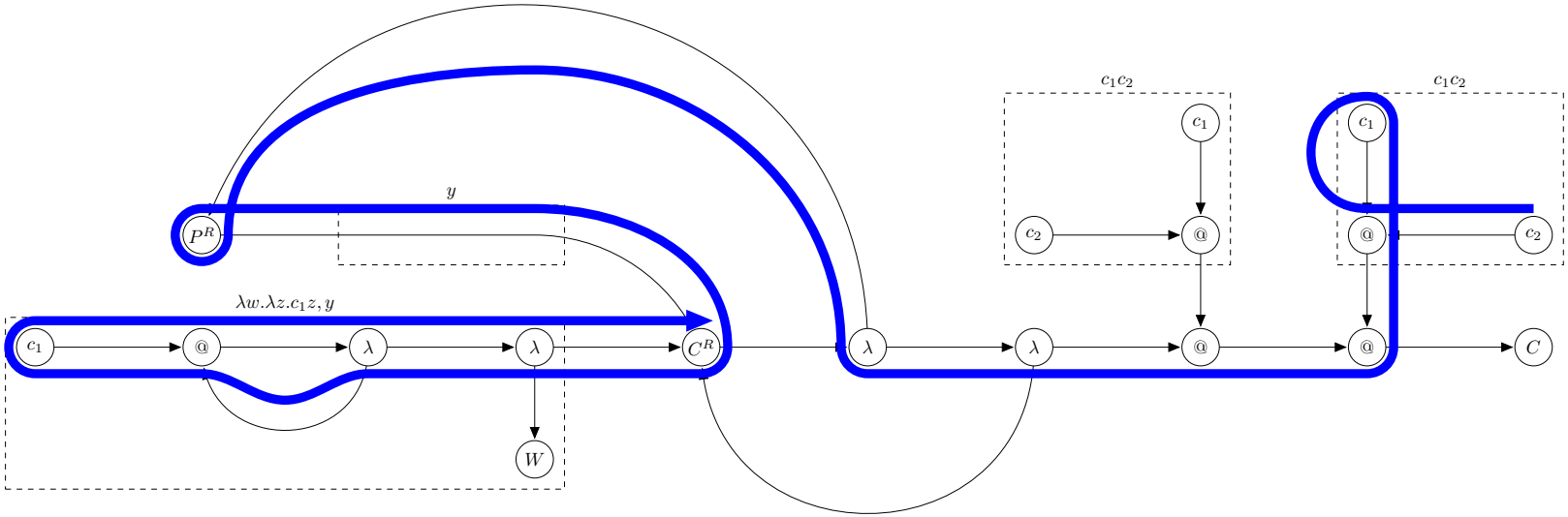
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



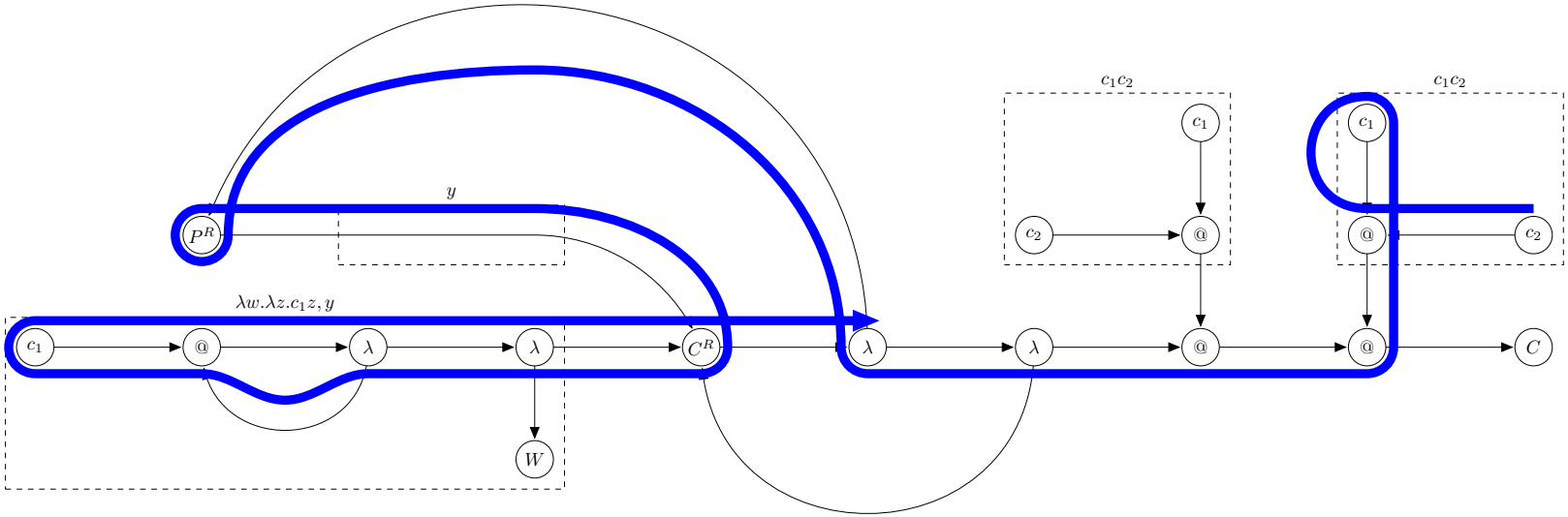
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



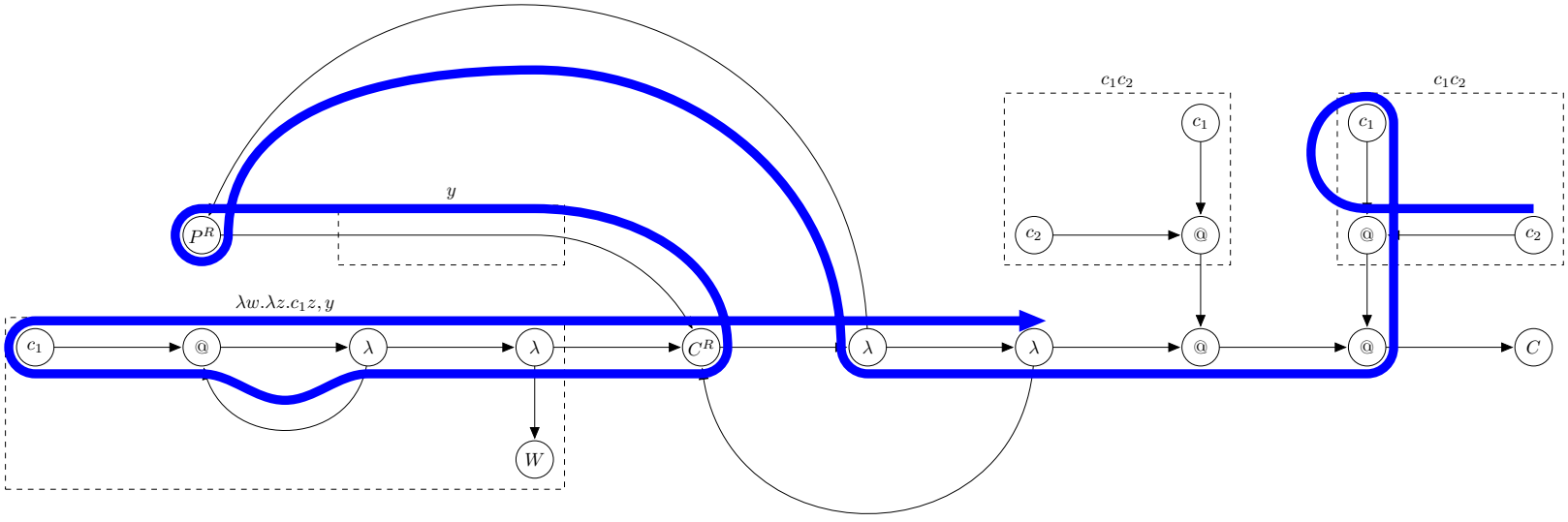
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



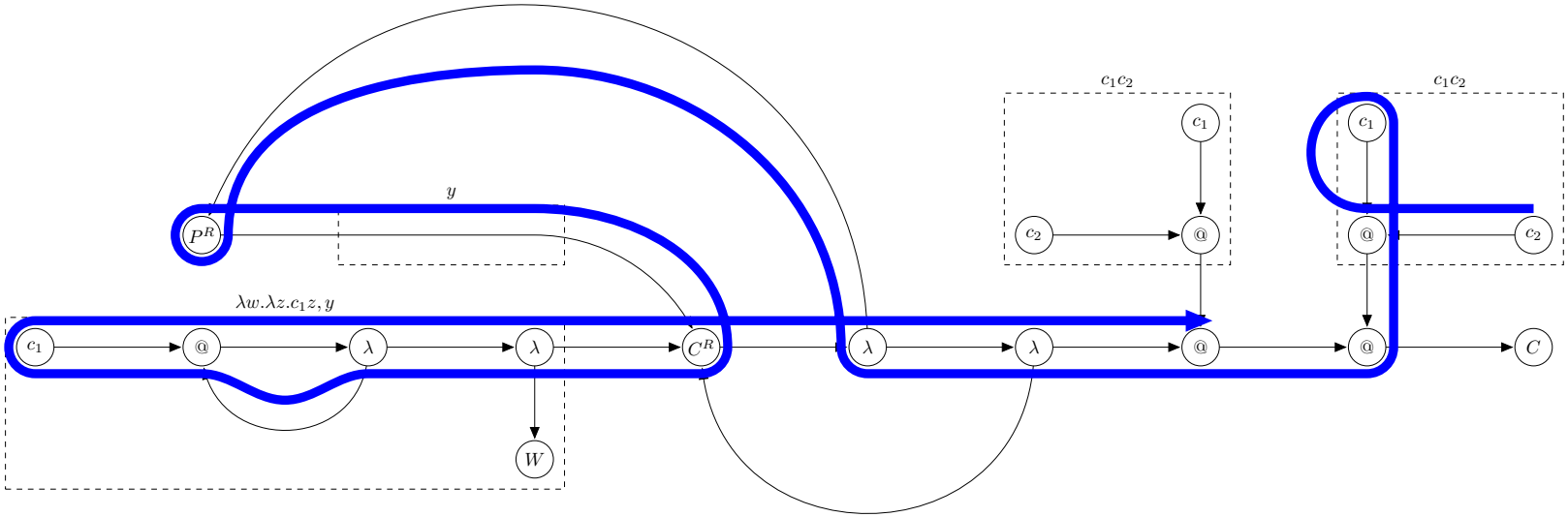
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



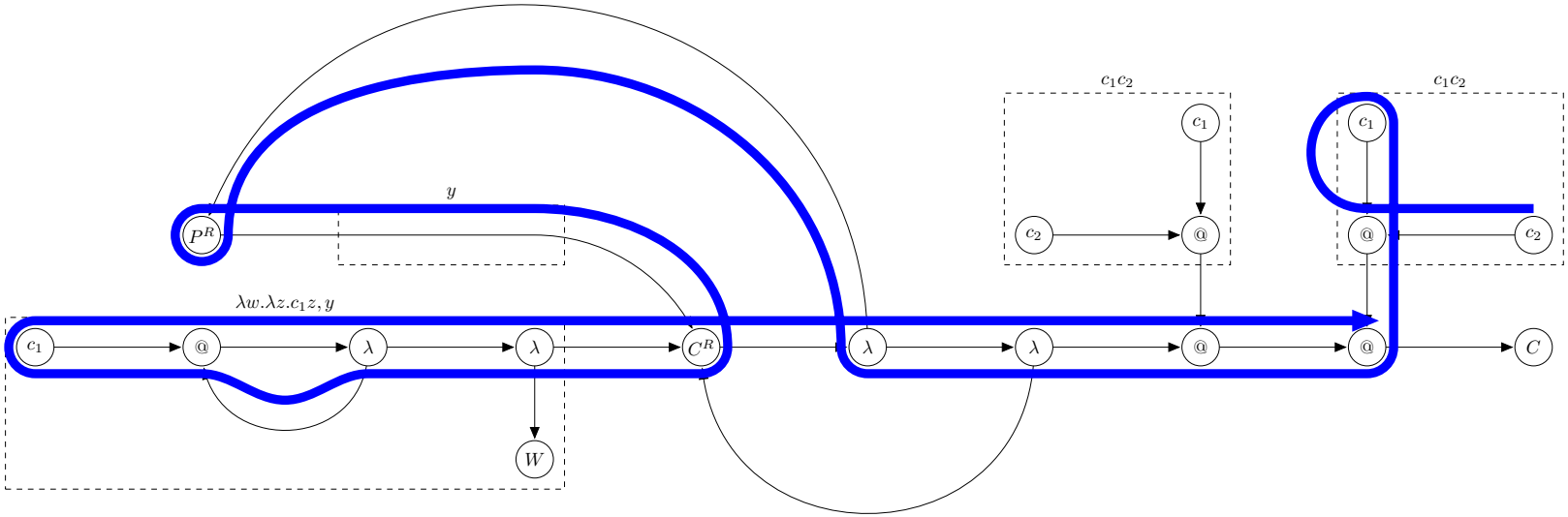
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



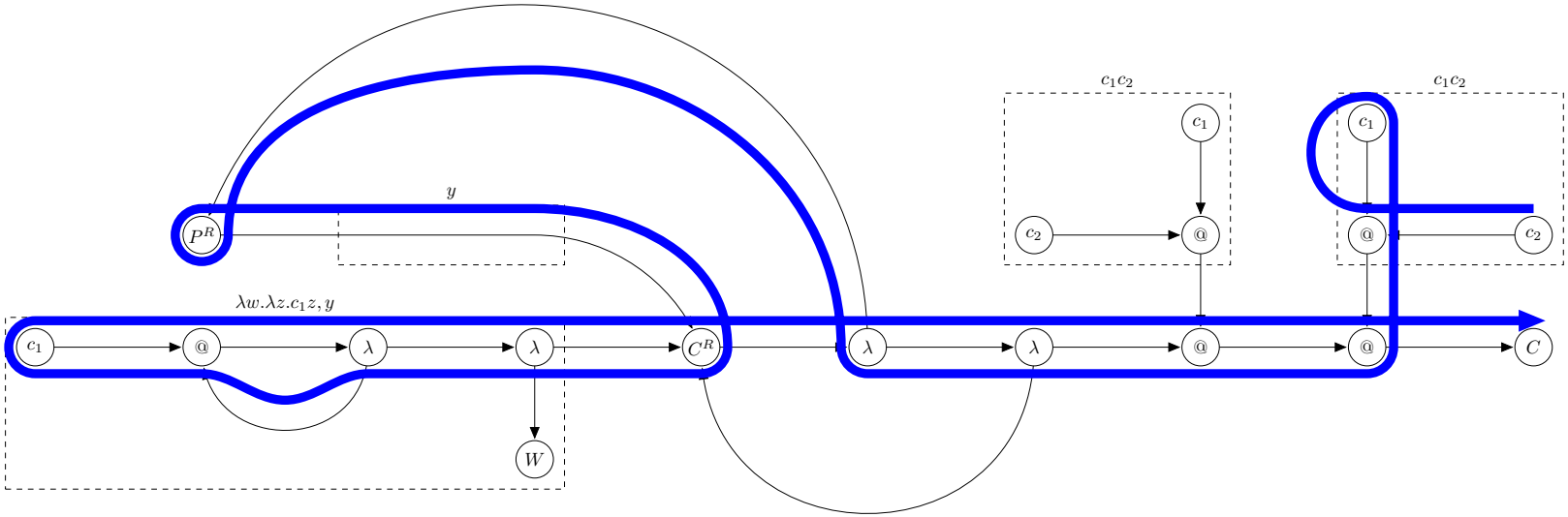
An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



An Example

$$(\lambda x.\lambda y.x\langle\langle\lambda w.\lambda z.c_1\, z, y\rangle\rangle)(c_1\, c_2)(c_1\, c_2)$$



Algebraic Context Semantics is Enough

- First of all, we can give the following

Lemma (Adequacy)

If $\pi : \Gamma \vdash M : A$ and M contains a redex with argument t, then there is $T \in \mathcal{T}(G_\pi)$ with $\Phi(T) = t$.

- Moreover:

Lemma (Backward Preservation)

If $\pi : \Gamma \vdash M : A$ and $M \rightsquigarrow N$, there is $\xi : \Gamma \vdash N : A$ such that whenever $T \in \mathcal{T}(G_\xi)$ there is $U \in \mathcal{T}(G_\pi)$ with $\Phi(T) = \Phi(U)$.

- As a consequence:

Theorem

If $\pi : \Gamma \vdash M : A$, then there is $T \in \mathcal{T}(G_\pi)$ with $|\Phi(T)| = A(M)$.

Algebraic Context Semantics is Enough

- First of all, we can give the following

Lemma (Adequacy)

If $\pi : \Gamma \vdash M : A$ and M contains a redex with argument t, then there is $T \in \mathcal{T}(G_\pi)$ with $\Phi(T) = t$.

- Moreover:

Lemma (Backward Preservation)

If $\pi : \Gamma \vdash M : A$ and $M \rightsquigarrow N$, there is $\xi : \Gamma \vdash N : A$ such that whenever $T \in \mathcal{T}(G_\xi)$ there is $U \in \mathcal{T}(G_\pi)$ with $\Phi(T) = \Phi(U)$.

- As a consequence:

Theorem

If $\pi : \Gamma \vdash M : A$, then there is $T \in \mathcal{T}(G_\pi)$ with $|\Phi(T)| = A(M)$.

Algebraic Context Semantics is Enough

- First of all, we can give the following

Lemma (Adequacy)

If $\pi : \Gamma \vdash M : A$ *and* M *contains a redex with argument* t*, then there is* $T \in \mathcal{T}(G_\pi)$ *with* $\Phi(T) = t$.

- Moreover:

Lemma (Backward Preservation)

If $\pi : \Gamma \vdash M : A$ *and* $M \rightsquigarrow N$*, there is* $\xi : \Gamma \vdash N : A$ *such that whenever* $T \in \mathcal{T}(G_\xi)$ *there is* $U \in \mathcal{T}(G_\pi)$ *with* $\Phi(T) = \Phi(U)$.

- As a consequence:

Theorem

If $\pi : \Gamma \vdash M : A$*, then there is* $T \in \mathcal{T}(G_\pi)$ *with* $|\Phi(T)| = A(M)$.

Keeping Everything Under Control

- First of all, there is a uniqueness property:

Lemma (Uniqueness)

For every context $(A[\cdot], C)$, there is at most one tree $T \in \mathcal{T}(G)$ whose root is labelled with $(A[\cdot], C)$.

As a consequence, any context can appear at most once in a given tree T.

- Moreover, for every context $(A[\cdot], C)$ appearing in a tree, the type of the corresponding edge is an instance of $A[\cdot]$.

Keeping Everything Under Control

- First of all, there is a uniqueness property:

Lemma (Uniqueness)

For every context $(A[\cdot], C)$, there is at most one tree $T \in \mathcal{T}(G)$ whose root is labelled with $(A[\cdot], C)$.

As a consequence, any context can appear at most once in a given tree T.

- Moreover, for every context $(A[\cdot], C)$ appearing in a tree, the type of the corresponding edge is an instance of $A[\cdot]$.

What Remains to be Done?

- We only need to analyze how restrictions to π reflect to $\mathcal{T}(G_\pi)$.
- In particular, we prove bounds on the size of elements of $\Phi(\mathcal{T}(G_\pi))$.
- Relatively easy, exploiting induction on trees.
- For every fragment, a few lemmas suffice.

Syntax

- Formulae:

$$A ::= \alpha \mid A \multimap A \mid A \otimes A \mid !A \mid \forall \alpha.A$$

- Rules:

$$\frac{}{A \vdash A}\ A \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ U \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B}\ W \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ X$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ R_{\multimap} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ L_{\multimap}$$

$$\frac{A_1, \ldots, A_n \vdash B}{!A_1, \ldots, !A_n \vdash !B}\ P_! \qquad \frac{A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ D_! \qquad \frac{!!A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ N_!$$

$$\frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \forall \alpha.A}\ R_{\forall} \qquad \frac{\Gamma, A\{B/\alpha\} \vdash C}{\Gamma, \forall \alpha.A \vdash C}\ L_{\forall}$$

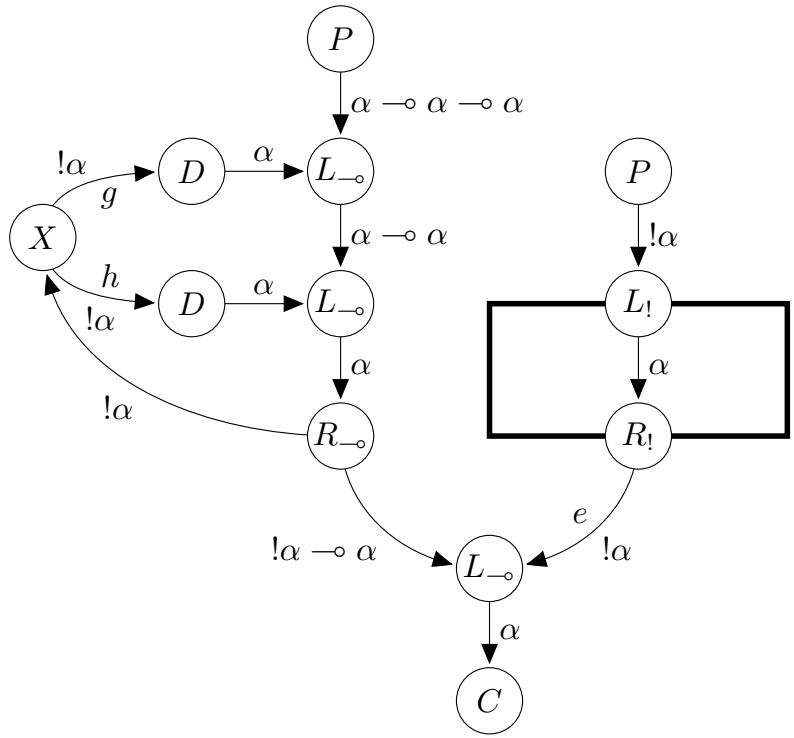
Proof-nets

- We adopt a system of intuitionistic proof-nets.
- For example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\,A \quad \dfrac{}{\alpha \vdash \alpha}\,A}{\alpha, \alpha \multimap \alpha \vdash \alpha}\,L_! \quad \dfrac{}{\alpha \vdash \alpha}\,A}{\alpha, \alpha, \alpha \multimap \alpha \multimap \alpha \vdash \alpha}\,L_!}{!\alpha, \alpha, \alpha \multimap \alpha \multimap \alpha \vdash \alpha}\,D_!}{!\alpha, !\alpha, \alpha \multimap \alpha \multimap \alpha \vdash \alpha}\,D_!}{!\alpha, \alpha \multimap \alpha \multimap \alpha \vdash \alpha}\,X}{\alpha \multimap \alpha \multimap \alpha \vdash !\alpha \multimap \alpha}\,R_! \qquad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,A \quad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,A}{!\alpha \vdash !\alpha}\,P_!}{!\alpha \multimap \alpha, !\alpha \vdash \alpha}\,L_!}{\alpha \multimap \alpha \multimap \alpha, !\alpha \vdash \alpha}\,U$$

Proof-nets

- We adopt a system of intuitionistic proof-nets.
- For example:



Copying - The Root of Complexity?

- Linear logic without exponentials and additives cannot express anything beyond polynomial time.
- Suppose we know the total number of times boxes in a proof-net G can (possibly) be duplicated. Call it W_G.
- Notice W_G can even be infinite.
- How W_G relates to the complexity of normalizing G?
- Context semantics helps in giving strong answers to this question.

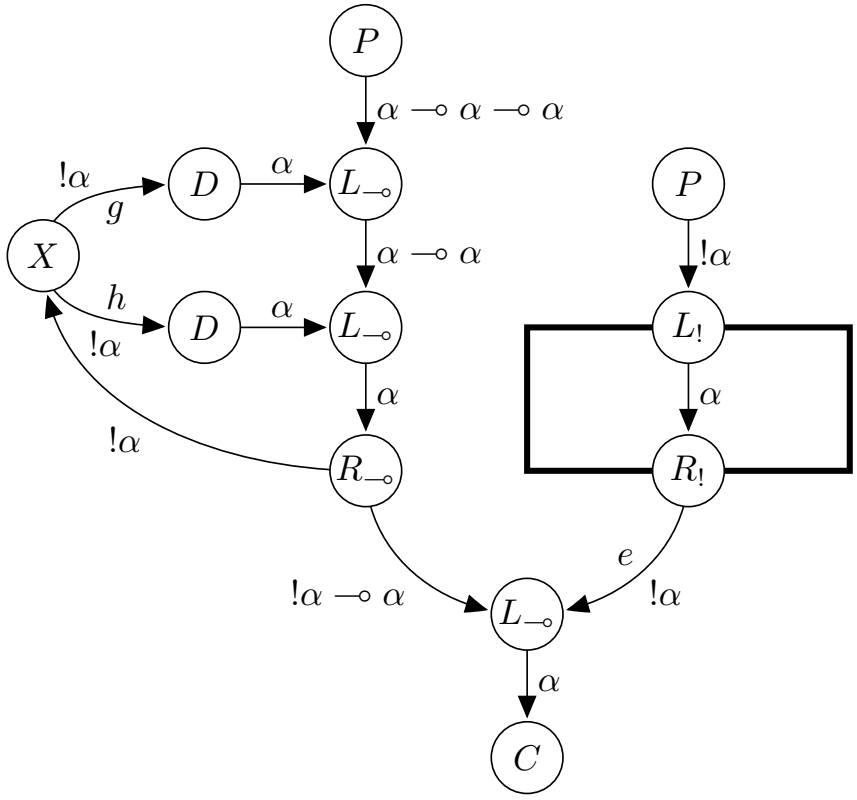
Context Semantics in Five Minutes

- Every proof-net G induces rewriting rules on

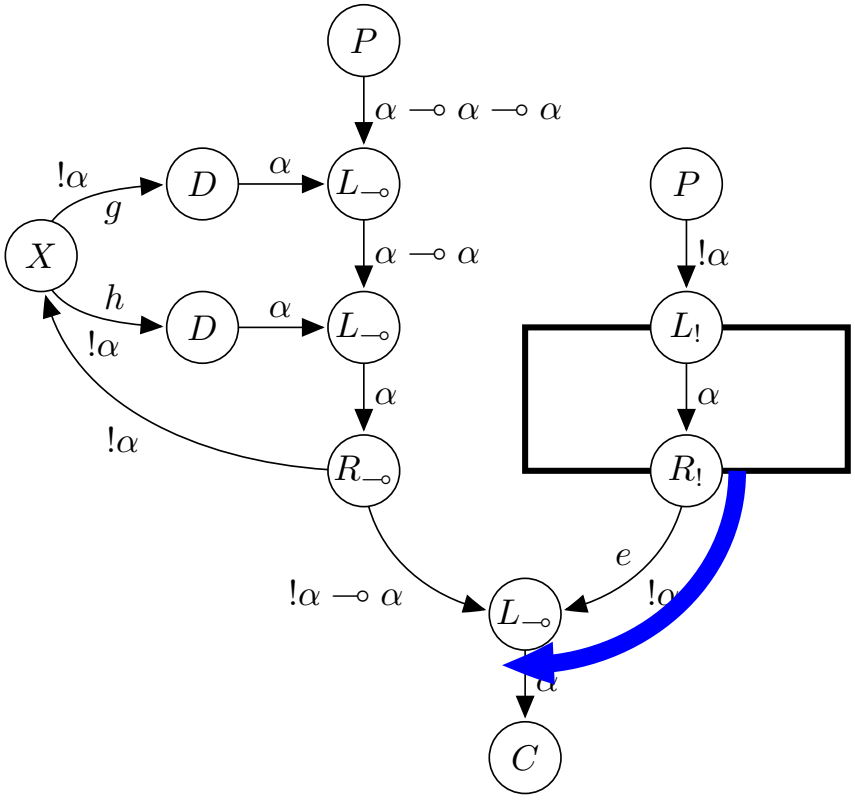
$$E_G \times C_G$$

  where E_G is the set of G edges, and C_G is a set of *contexts* for G.
- If $(e_1, C_1) \to (e_2, C_2) \to \ldots \to (e_n, C_n)$, then $e_1, \ldots, e_n$ is a *persistent path*.
- There are certain contexts called *exponential trees*.
- If e is a box main premise, then every exponential tree t with $(e, t) \to \ldots \to (g, u)$ (where u is somehow minimal) corresponds to a different "copy" of the box.
- W_G as the sum over all boxes of the number of exponential trees satisfying the above condition.

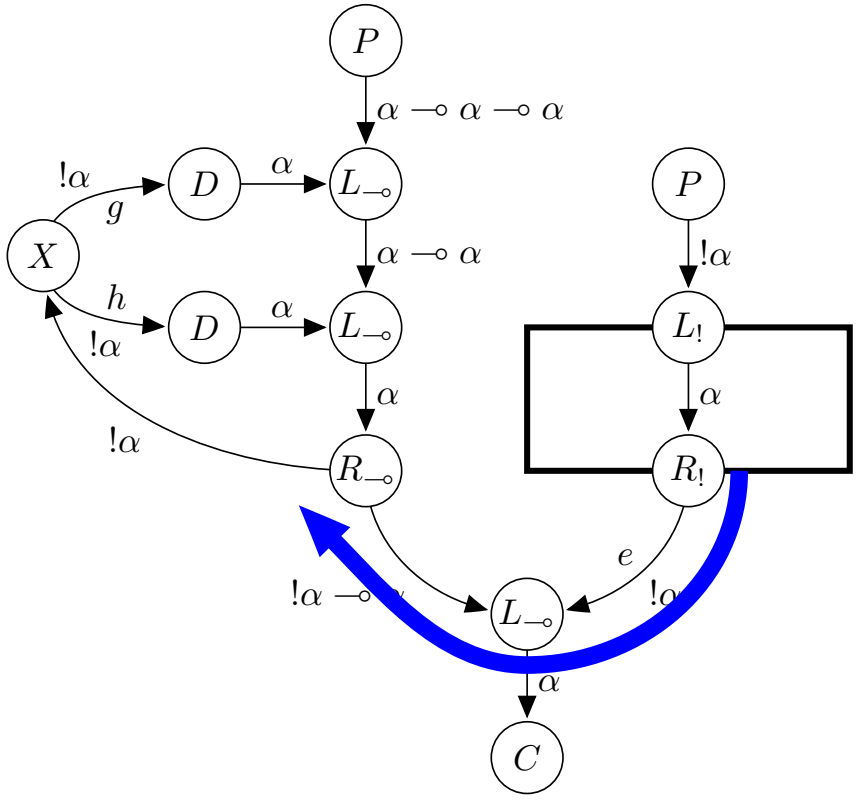
An Example



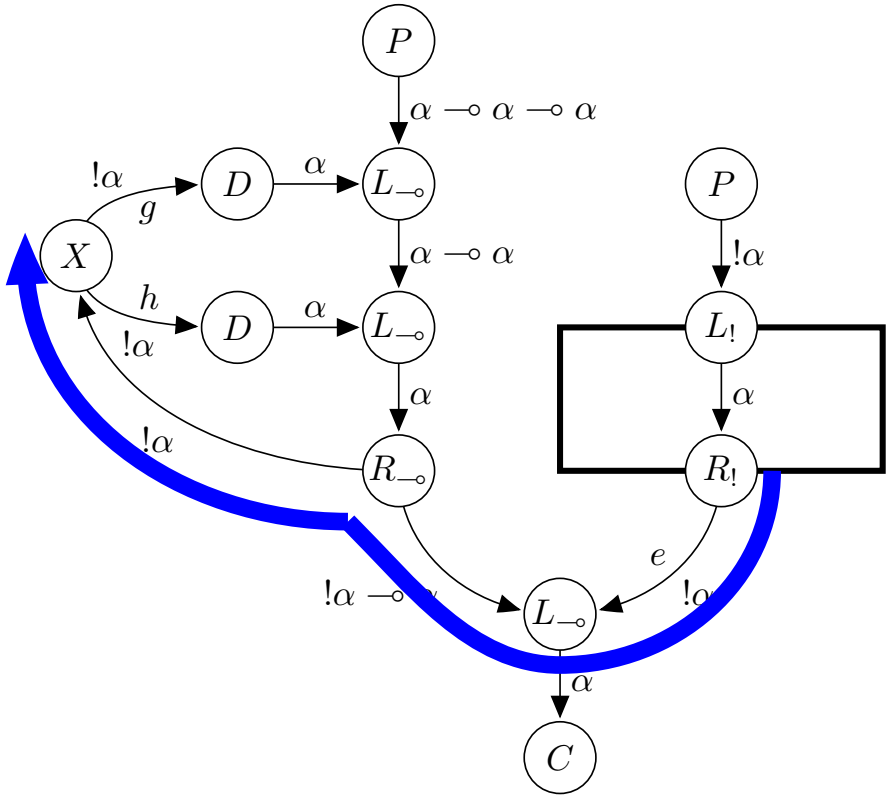
An Example



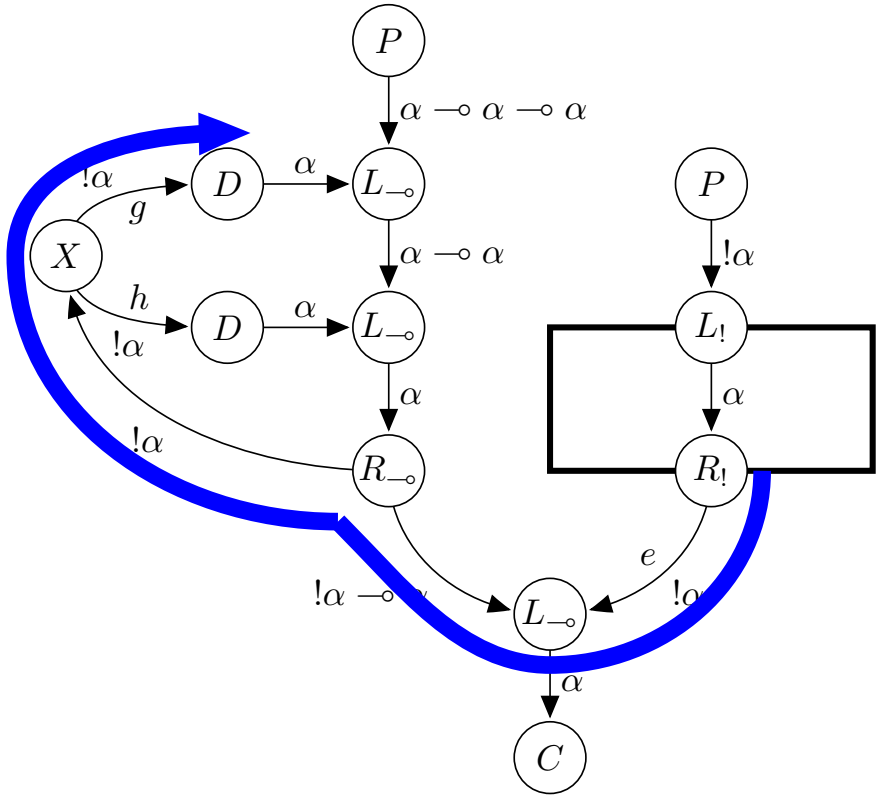
An Example



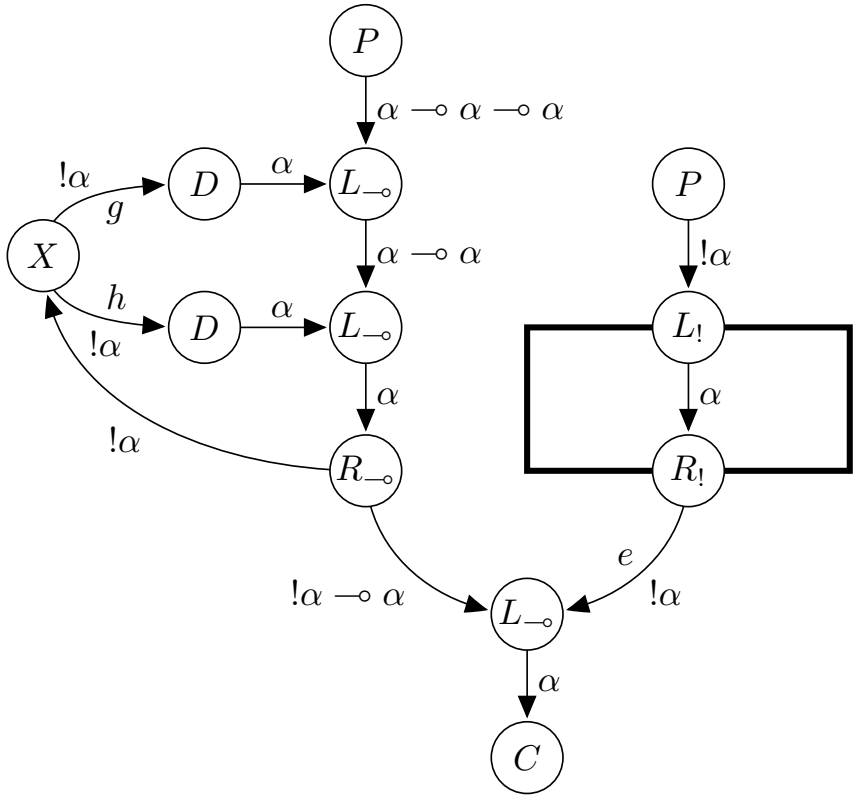
An Example



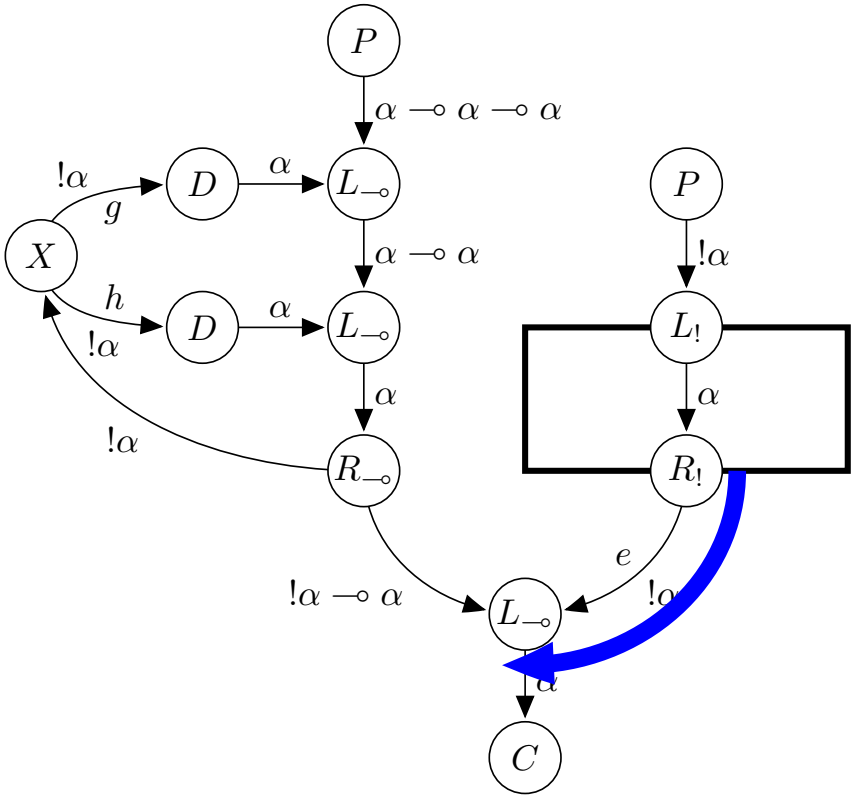
An Example



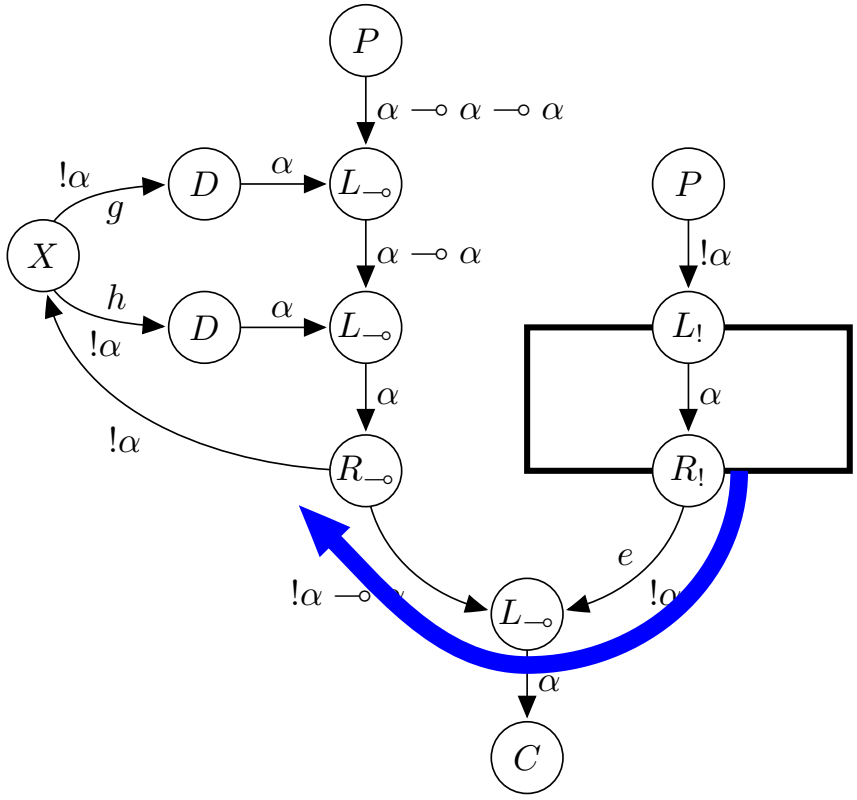
An Example



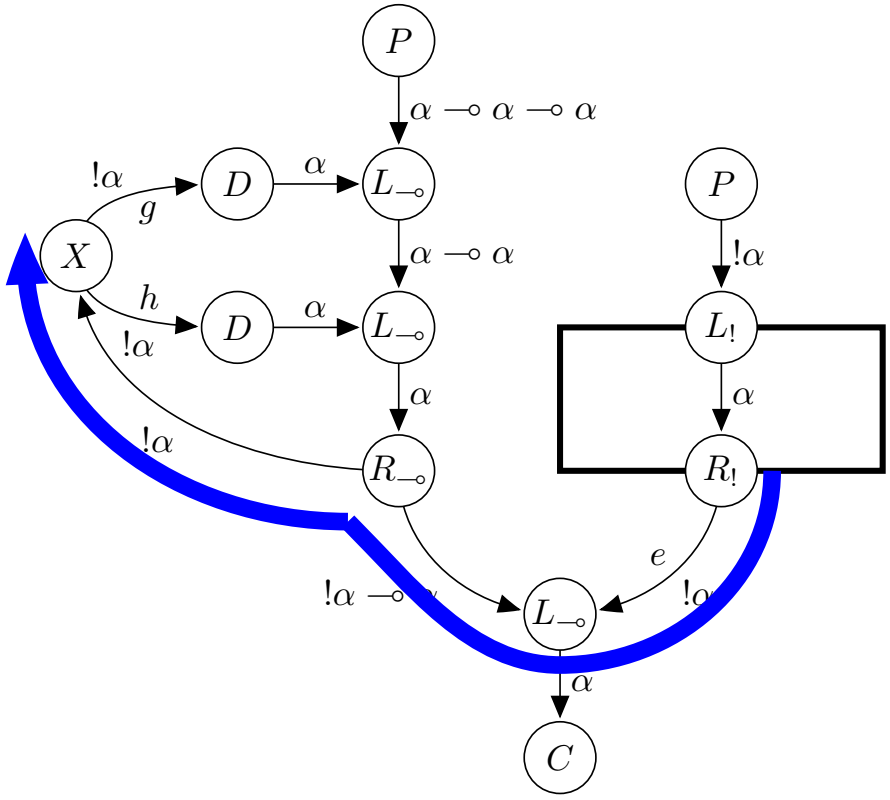
An Example



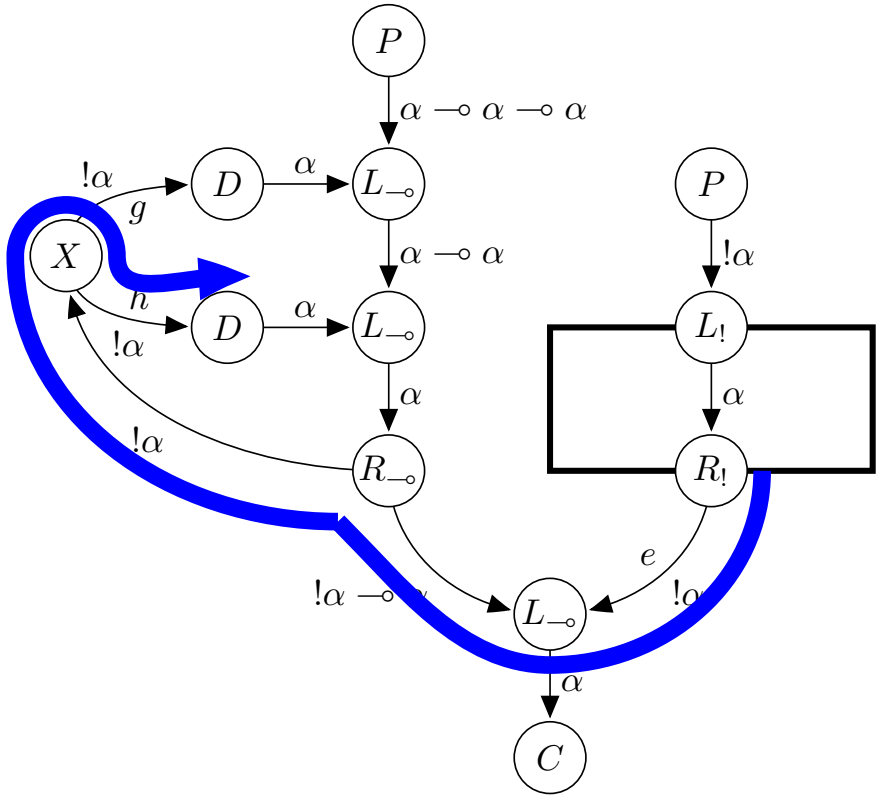
An Example



An Example



An Example



One Direction

- W_G is always positive, but is not guaranteed to strictly decrease at any normalization step.
- Moreover, it is not clear whether the size of $|G|$ is related to W_G or not.
- However, we can define from W_G and $|G|$ another quantity T_G which *both* decreases at every normalizatop step and majorizes $|G|$.
- As a consequence, we get:

Theorem

There is a polynomial $p : \mathbb{N}^2 \to \mathbb{N}$ such that for every proof-net G, G normalizes in at most $p(W_G, |G|)$ steps and the size of any reduct H of G is at most $p(W_G, |G|)$.

The Other Direction

- Is W_G a gross overestimate on the time needed to normalize G?
- Consider the *level-by-level* strategy:
  - For every $n \in \mathbb{N}$, a cut at level $n+1$ is fired only if there are not cuts at levels from 1 to n.
  - For every $n \in \mathbb{N}$, a !-cut at level n is fired only if every cut at level n is either a W-cut or a !-cut
  - A W-cut is fired only if every cut in the proof-net is a W-cut.
- W_G decreases by **at most one** at every step.
- As a consequence, we get:

Theorem

Let G be a proof-net. If $W_G = \omega$, then G diverges, otherwise there is H with $G \rightarrow^{W_G} H$.

Contributions

- We have proved **several charaterization results** for the class of representable functions in fragments of higher-order recursion.
- Most work is **factored over** the various fragments and done just once.
- Copying has been proved to be **the real source** of complexity in the realm of linear logic.

Contributions

- We have proved **several charaterization results** for the class of representable functions in fragments of higher-order recursion.
- Most work is **factored over** the various fragments and done just once.
- Copying has been proved to be **the real source** of complexity in the realm of linear logic.

Future Developments

- Other fragments of higher-order recursion (e.g. Non-size increasing polytime computation) seem to be analyzable in our framework.
- An interesting development consists in applying the methodology to Mackie's interaction nets for PCF.
- (Re)prove soundness theorems for linear logic fragments by way of context semantics.
- Is context semantics related to resource polynomials and weights in bounded linear logic?

Questions?