Coalgebras in Dependent Type Theory

Anton Setzer (Swansea), Peter Hancock (Edinburgh)

Coalgebras in Dependent Type Theory

Anton Setzer (Swansea), Peter Hancock (Edinburgh)

1. Interactive Programs and why we need Coalgebras.

Coalgebras in Dependent Type Theory

Anton Setzer (Swansea), Peter Hancock (Edinburgh)

1. Interactive Programs and why we need Coalgebras.

2. Rules for Coalgebras.

Coalgebras in Dependent Type Theory

Anton Setzer (Swansea), Peter Hancock (Edinburgh)

1. Interactive Programs and why we need Coalgebras.

2. Rules for Coalgebras.

3. Some Operations on Coalgebras.

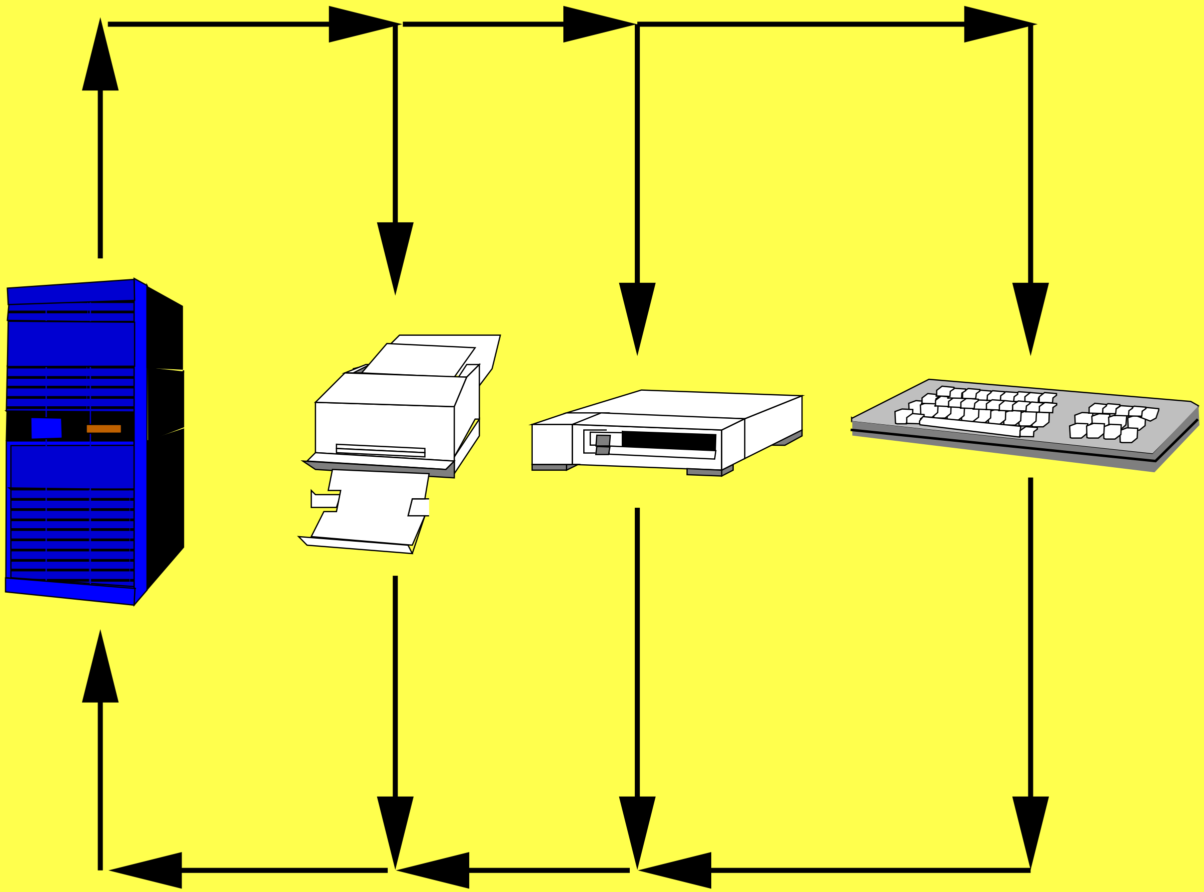
Coalgebras in Dependent Type Theory

Anton Setzer (Swansea), Peter Hancock (Edinburgh)

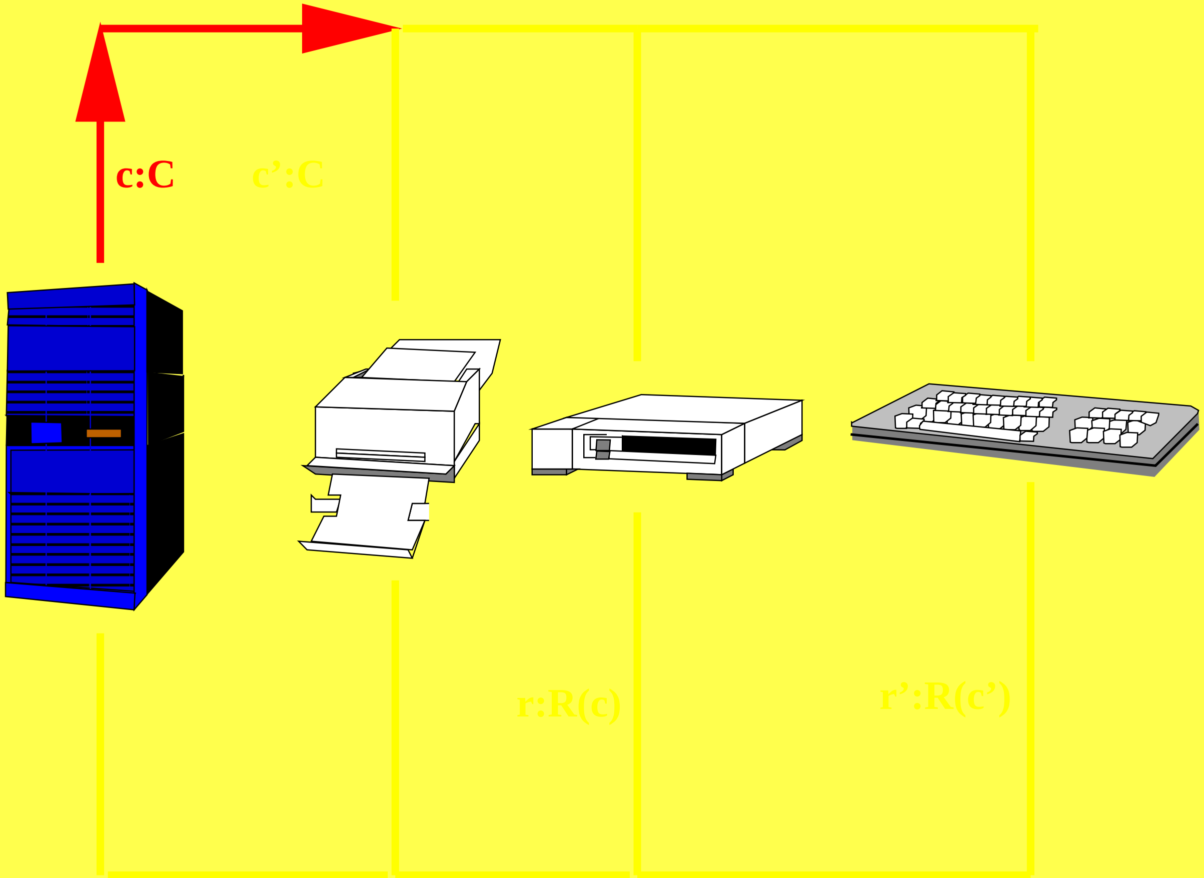
1. Interactive Programs and why we need Coalgebras.
2. Rules for Coalgebras.
3. Some Operations on Coalgebras.
4. The μ-Operator and Coalgebras.

1. Interactive Programs and why we need Coalgebras

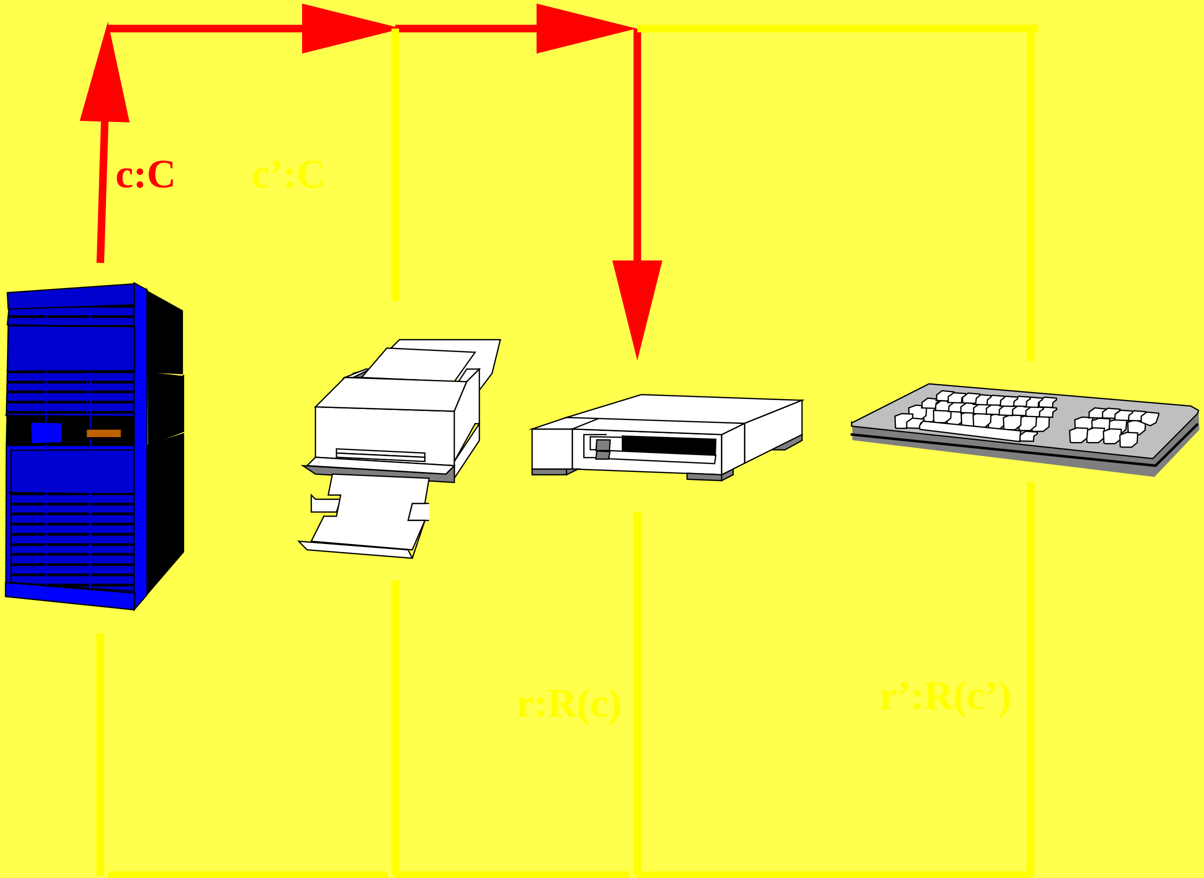
1. Interactive Programs and why we need Coalgebras



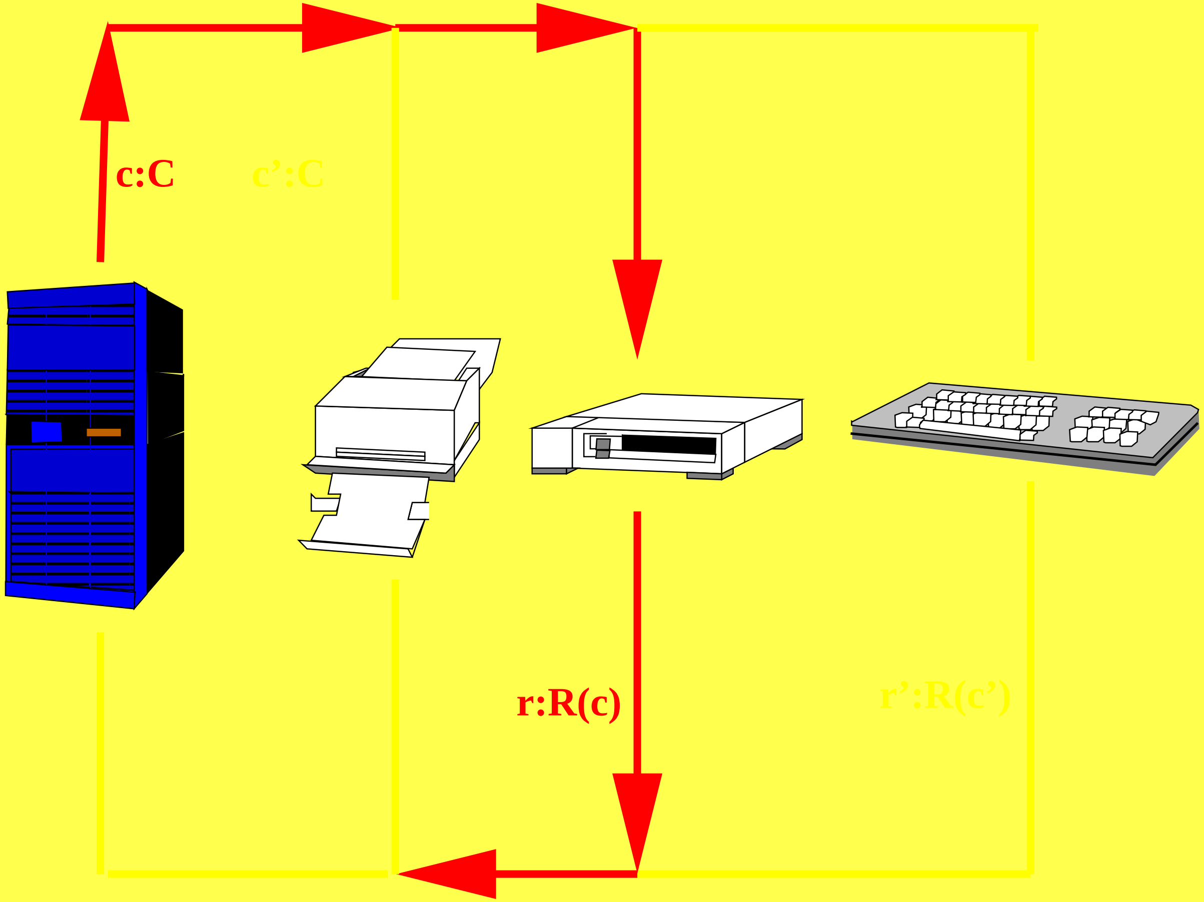
1. Interactive Programs and why we need Coalgebras



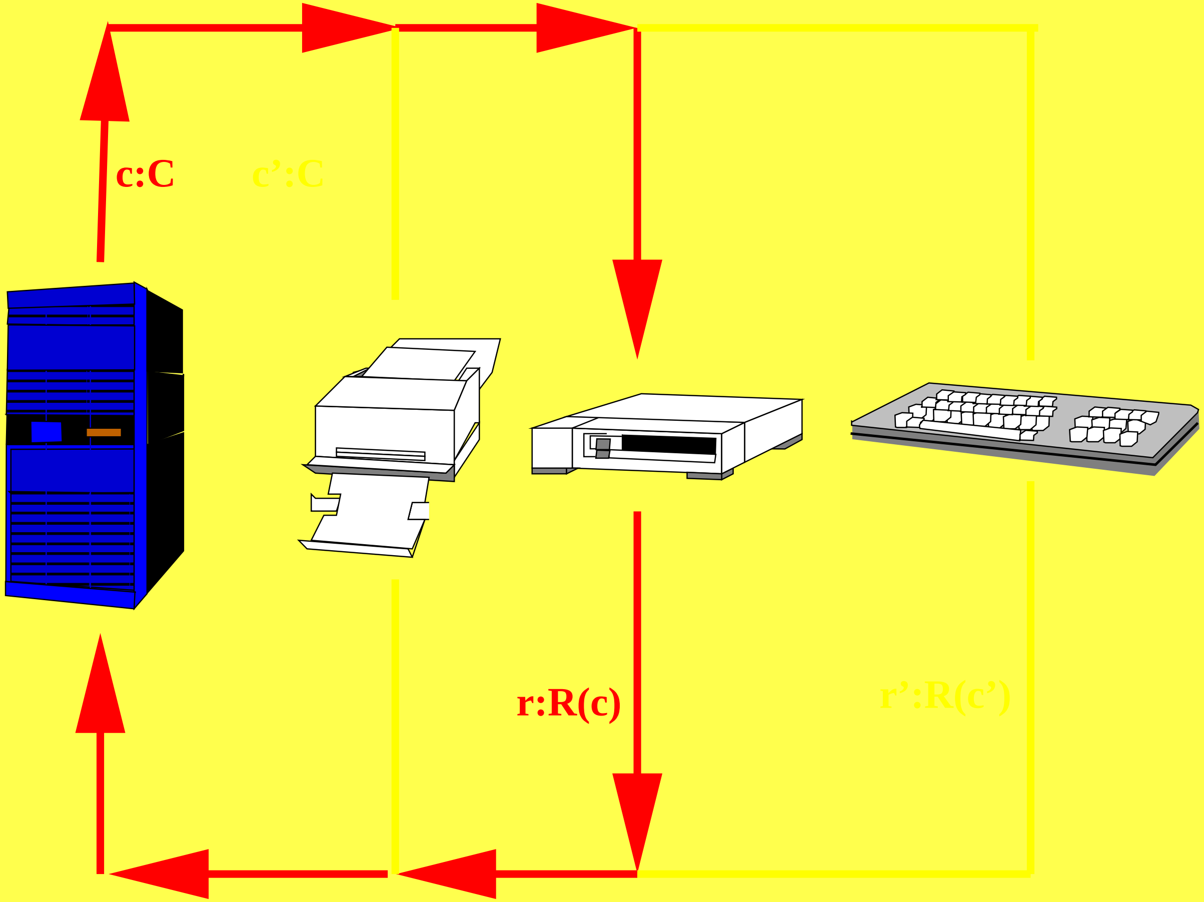
1. Interactive Programs and why we need Coalgebras



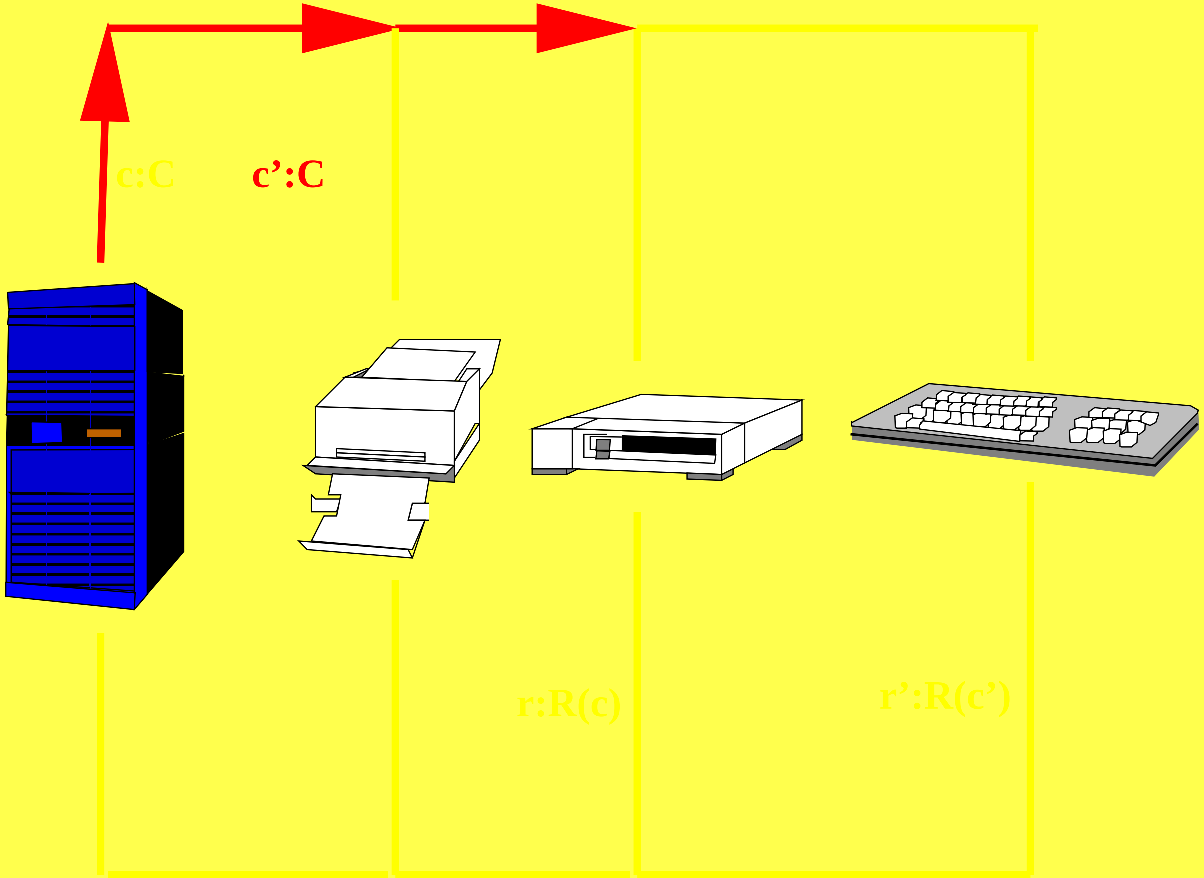
1. Interactive Programs and why we need Coalgebras



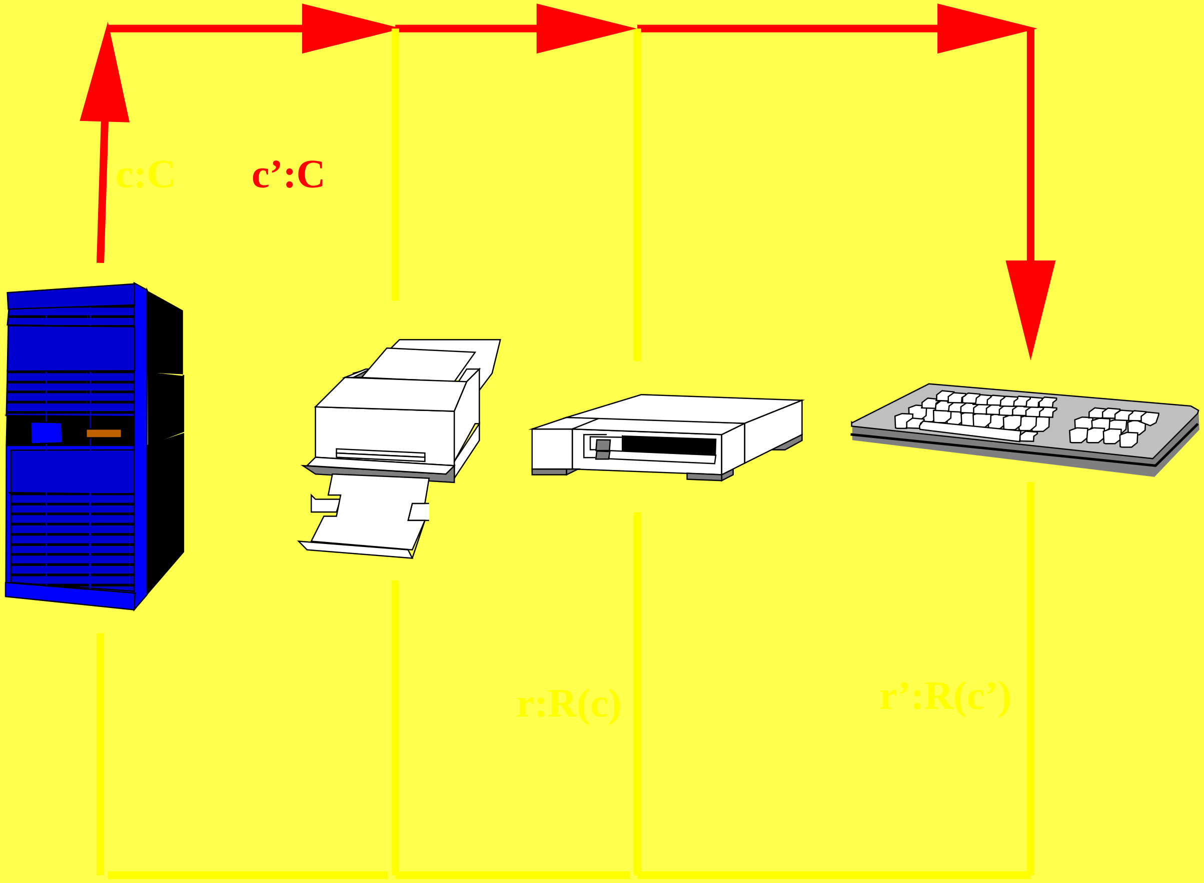
1. Interactive Programs and why we need Coalgebras



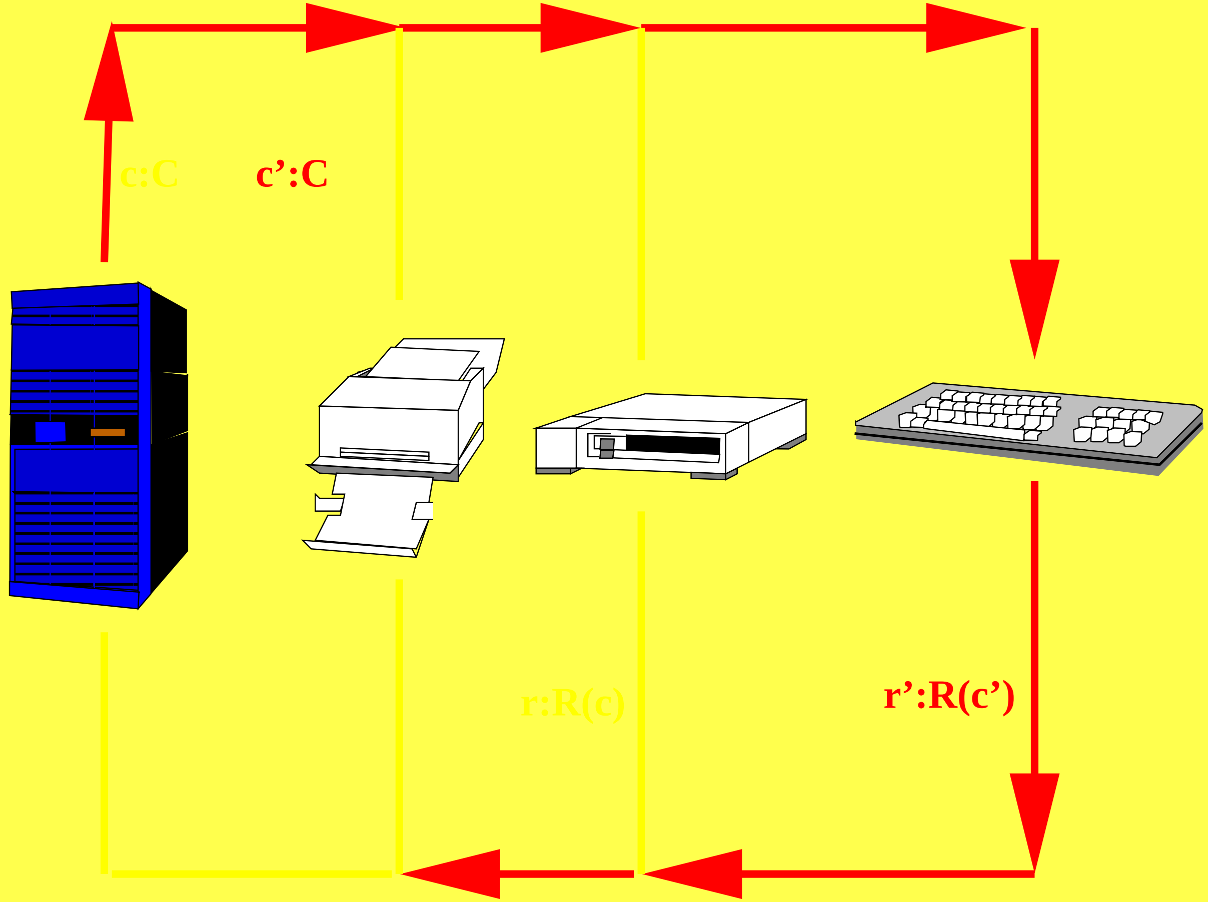
1. Interactive Programs and why we need Coalgebras



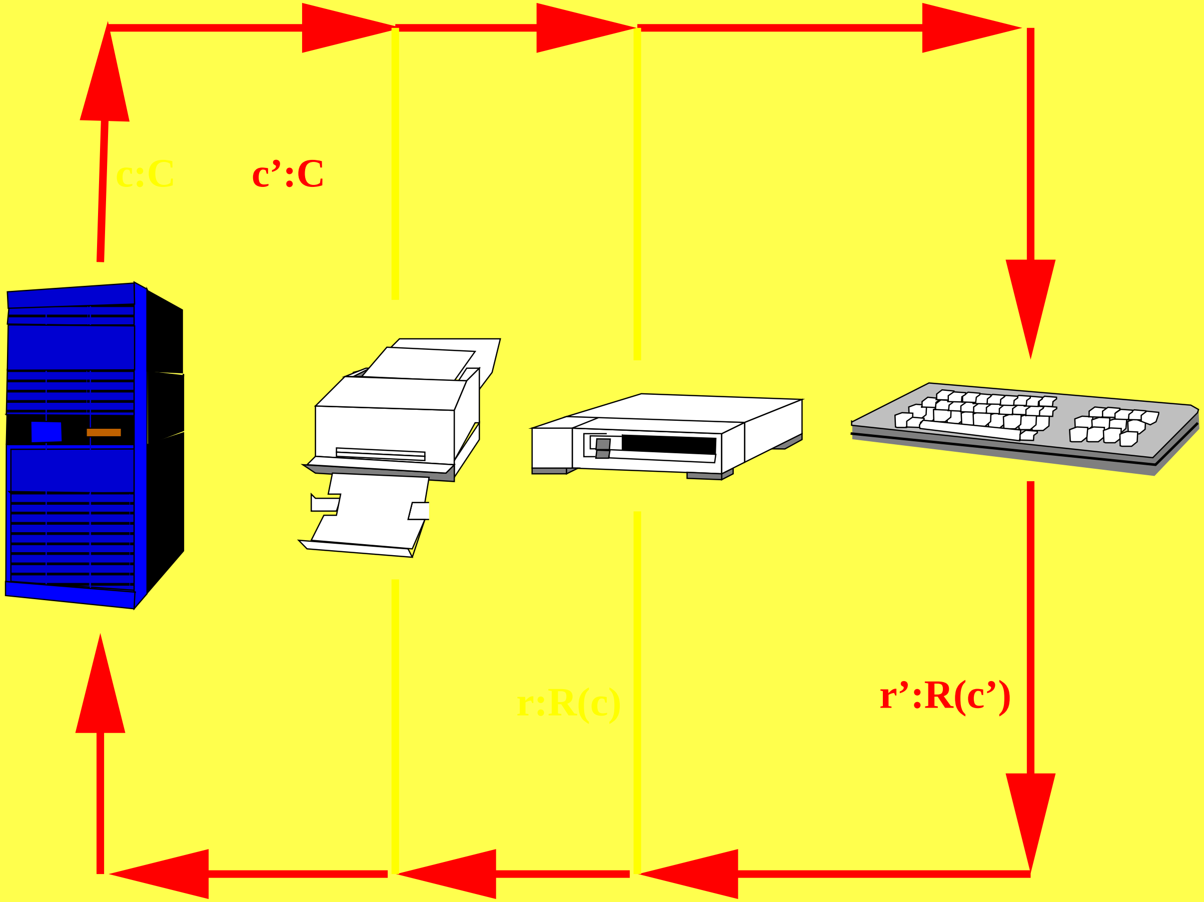
1. Interactive Programs and why we need Coalgebras



1. Interactive Programs and why we need Coalgebras



1. Interactive Programs and why we need Coalgebras



Representation of Interactive Programs: IO-Trees

Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)

Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

- IO = set of non-well-founded trees with

Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

- $\underset{\sim}{\mathrm{IO}}$ = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,

Representation of Interactive Programs: IO-Trees

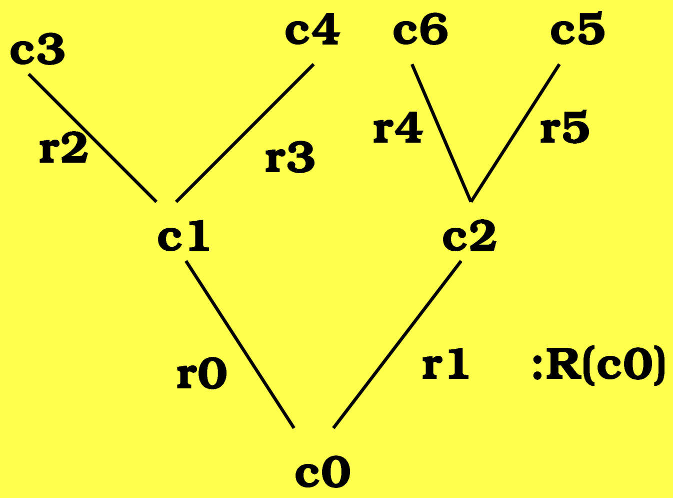
- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$

Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

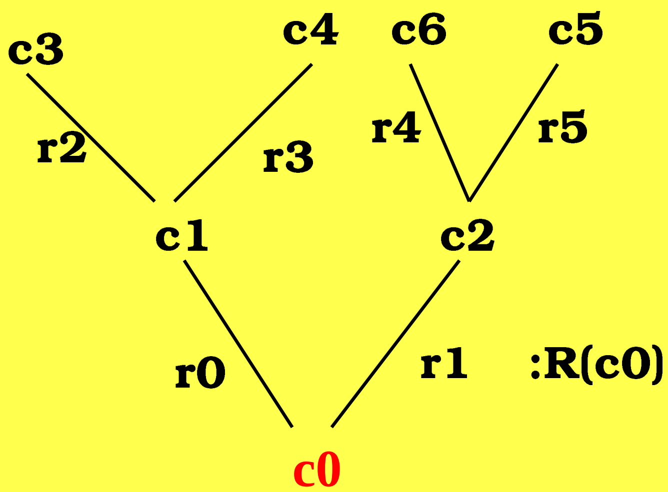
- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

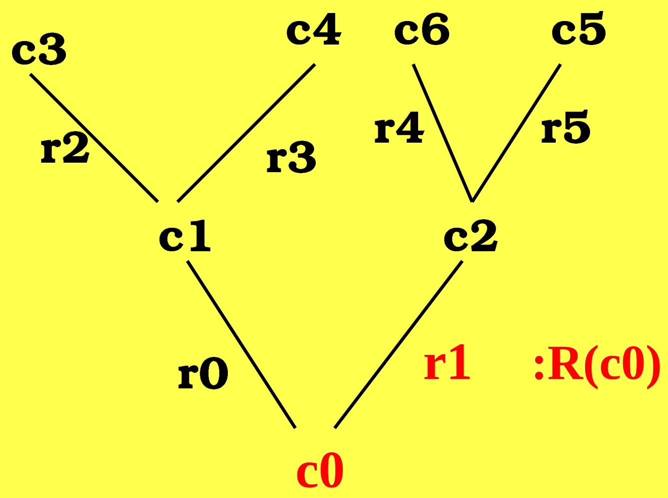
- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

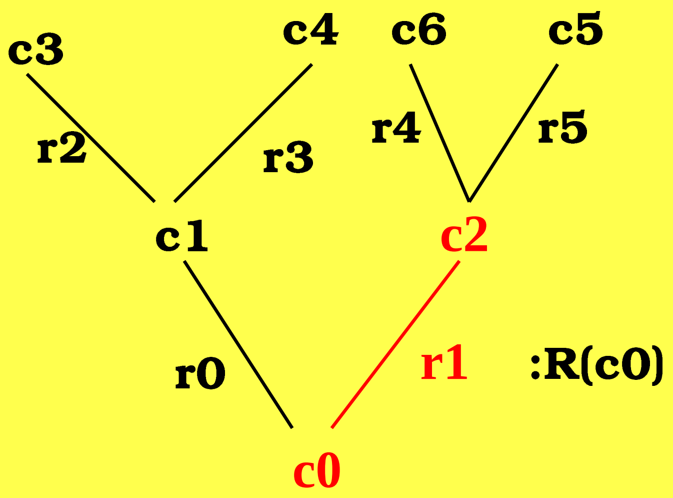
- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

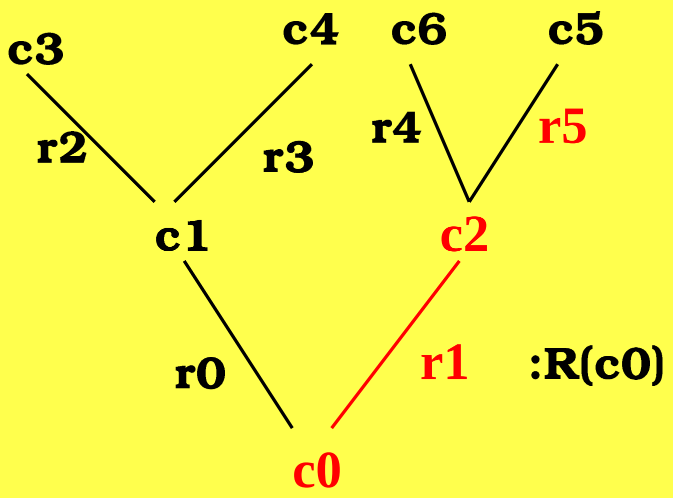
- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

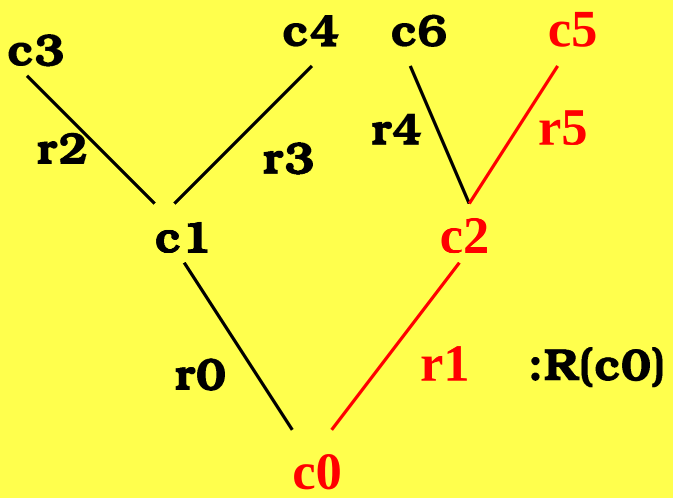
- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Representation of Interactive Programs: IO-Trees

- **Assume**
  - $C : \mathrm{Set}$ (set of commands)
  - $R(c) : \mathrm{Set}$ for $c : C$ (set of responses for command c).

- IO = set of non-well-founded trees with
  - nodes **labeled** by $c : C$,
  - node with label c has **branching degree** $R(c)$



Problem

Problem

- What do we mean by the set of **non-well-founded trees**?

Problem

- What do we mean by the set of **non-well-founded trees**?
- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

Problem

- What do we mean by the set of **non-well-founded trees**?

- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

$\Rightarrow$ Need for representation of **coinductive** data types.

Problem

- What do we mean by the set of **non-well-founded trees**?
- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

$\Rightarrow$ Need for representation of **coinductive** data types.

- If IO is defined, we will have a function

$$\mathrm{elim} : \mathrm{IO} \to (\Sigma c : C.R(c) \to \mathrm{IO})$$

Problem

- What do we mean by the set of **non-well-founded trees**?
- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

$\Rightarrow$ Need for representation of **coinductive** data types.

- If IO is defined, we will have a function

$$\mathrm{elim} : \mathrm{IO} \to (\Sigma c : C.R(c) \to \mathrm{IO})$$

Problem

- What do we mean by the set of **non-well-founded trees**?

- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

$\Rightarrow$ Need for representation of **coinductive** data types.

- If IO is defined, we will have a function

$$\mathrm{elim} : \mathrm{IO} \to \underbrace{(\Sigma c : C.R(c) \to \mathrm{IO})}_{F(\mathrm{IO})} \; ,$$

Problem

- What do we mean by the set of **non-well-founded trees**?
- In predicative dependent type theory, only **inductive** data types available.
  - Only well-founded trees directly definable.

$\Rightarrow$ Need for representation of **coinductive** data types.

- If IO is defined, we will have a function

$$\mathrm{elim} : \mathrm{IO} \to \underbrace{(\Sigma c : C.R(c) \to \mathrm{IO})}_{F(\mathrm{IO})} ,$$
$$F := \lambda X.\Sigma c : C.R(c) \to X$$

Generalization

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.
  - If F, G are isomorphic to it, so is $\lambda X.F(X) + G(X)$.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.
  - If F, G are isomorphic to it, so is $\lambda X.F(X) + G(X)$.
  - If $A : \mathrm{Set}$, F_a is isomorphic to it ($a : A$), so are
    * $\lambda X.\Sigma a : A.F_a(X)$.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.
  - If F, G are isomorphic to it, so is $\lambda X.F(X) + G(X)$.
  - If $A : \mathrm{Set}$, F_a is isomorphic to it ($a : A$), so are
    - $\lambda X.\Sigma a : A.F_a(X)$.
    - $\lambda X.\Pi a : A.F_a(X)$ (use of axiom of choice).

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.
  - If F, G are isomorphic to it, so is $\lambda X.F(X) + G(X)$.
  - If $A : \mathrm{Set}$, F_a is isomorphic to it ($a : A$), so are
    * $\lambda X.\Sigma a : A.F_a(X)$.
    * $\lambda X.\Pi a : A.F_a(X)$ (use of axiom of choice).
- Call such operations **strictly positive functors**.

Generalization

- Many functions $F : \mathrm{Set} \to \mathrm{Set}$ isomorphic to $\lambda X.\Sigma c : C.R(c) \to X$ for some C, R.
  - $\lambda X.X$.
  - $\lambda X.C$.
  - If F, G are isomorphic to it, so is $\lambda X.F(X) + G(X)$.
  - If $A : \mathrm{Set}$, F_a is isomorphic to it ($a : A$), so are
    * $\lambda X.\Sigma a : A.F_a(X)$.
    * $\lambda X.\Pi a : A.F_a(X)$ (use of axiom of choice).
- Call such operations **strictly positive functors**.
- Notion could be extended to include F^* (inital algebra functor) and F^∞ (final coalgebra functor; see below) for F strictly positive.

Operation on Morphisms

- Operation on morphisms for $F = \lambda X.\Sigma c : C.R(c) \rightarrow X$:

Operation on Morphisms

- Operation on morphisms for $F = \lambda X.\Sigma c : C.R(c) \to X$:
  - If $f : X \to Y$, $F(f) : F(X) \to F(Y)$,

$$F(f)(\langle c, n \rangle) = \langle c, f \circ n \rangle \ .$$

Notation

$$\underset{\sim\sim\sim}{C_0(A) + C_1(B)} := \mathrm{data}\{C_0(a : A) \mid C_1(b : B)\}$$

2. Rules for Coalgebras.

Let F be strictly positive.
We need rules expressing

2. Rules for Coalgebras.

Let F be strictly positive.
We need rules expressing

- $\underset{\sim\sim\sim}{F_0^\infty}$

2. Rules for Coalgebras.

Let F be strictly positive.
We need rules expressing

- $\underset{\sim}{F_0^\infty}$ is (semi-) largest set s.t. there exists

$$\mathbf{elim} : F_0^\infty \to F(F_0^\infty) \ .$$

(F strictly positive).

2. Rules for Coalgebras.

Let F be strictly positive.
We need rules expressing

- F_0^∞ is (semi-) largest set s.t. there exists

$$\mathbf{elim} : F_0^\infty \to F(F_0^\infty) \ .$$

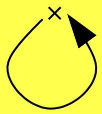
  (F strictly positive).

- Idea from Peter Aczel, **non-well-founded set theory:** Elements introduced as graphs.

Examples of Non-Wf Sets

$\{\{\{\cdots\}\}\}$ given by

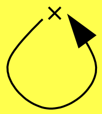
Examples of Non-Wf Sets

$\{\{\{\cdots\}\}\}$ given by 

or

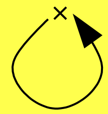
Examples of Non-Wf Sets

$\{\{\{\cdots\}\}\}$ given by

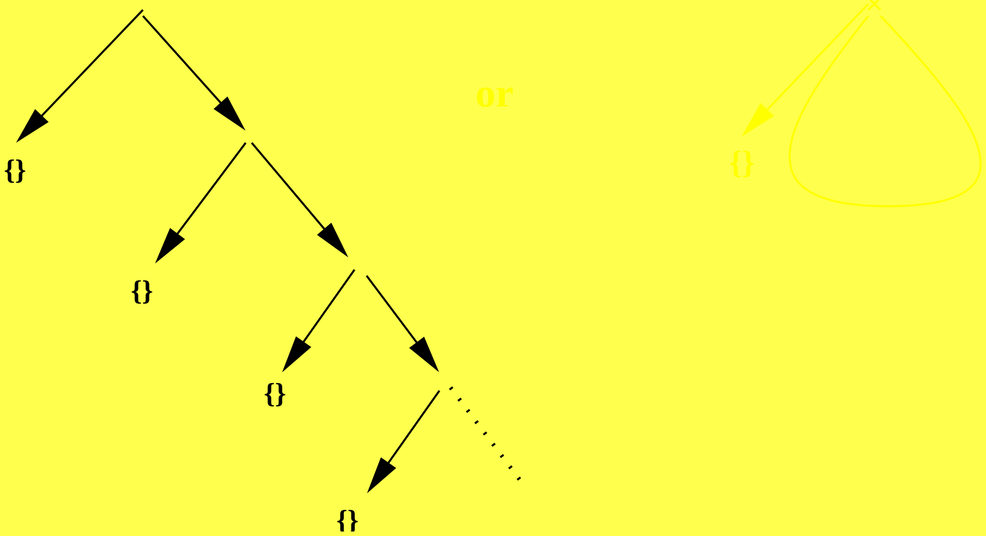


$\{\{\}\{\{\}\{\{\}\cdots\}\}\}$ given by

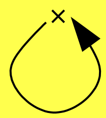
Examples of Non-Wf Sets

$\{\{\{\cdots\}\}\}$ given by 

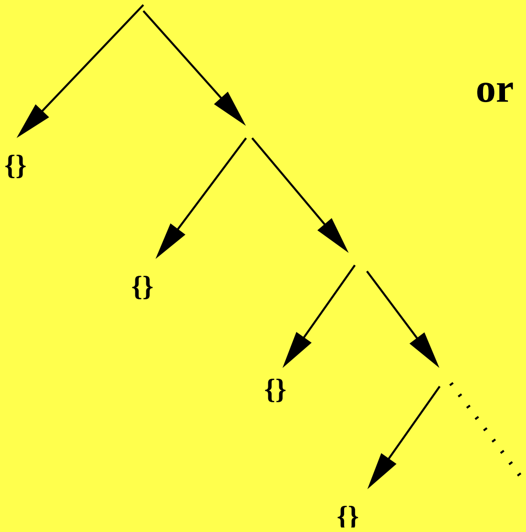
$\{\{\}\{\{\}\{\{\}\cdots\}\}\}$ given by



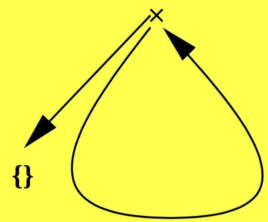
Examples of Non-Wf Sets

$\{\{\{\cdots\}\}\}$ given by 

$\{\{\}\{\{\}\{\{\}\cdots\}\}\}$ given by



or



Graphs for F_0^∞

Assume $F(X) = \Sigma c : C.R(c) \to X$.

Graphs for F_0^∞

Assume $F(X) = \Sigma c : C.R(c) \to X$.

- A **graph** for F consists of

Graphs for F_0^∞

Assume $F(X) = \Sigma c : C.R(c) \to X$.

- A **graph** for F consists of
  - a set A,

Graphs for F_0^∞

Assume $F(X) = \Sigma c : C.R(c) \to X$.

- A **graph** for F consists of
  - a set A,
  - a labelling function $c : A \to C$,

Graphs for F_0^∞

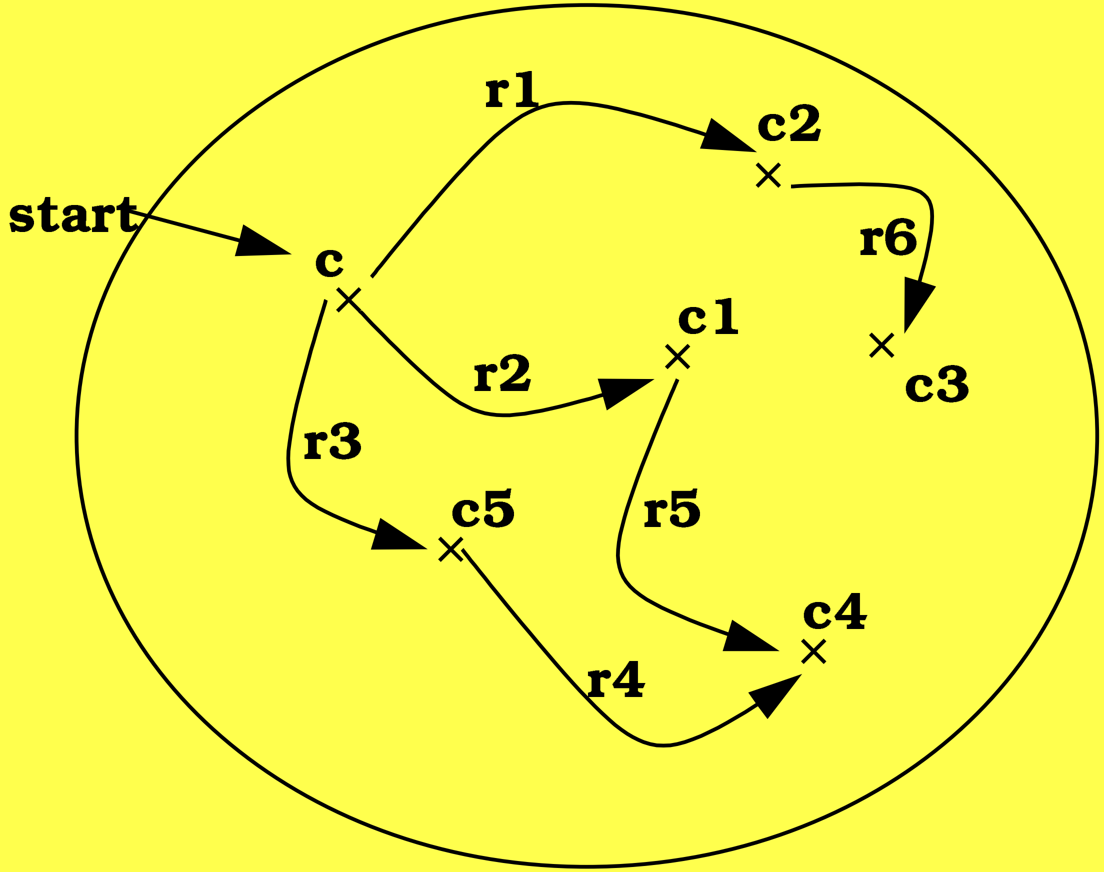
Assume $F(X) = \Sigma c : C.R(c) \to X$.

- A **graph** for F consists of
  - a set A,
  - a labelling function $c : A \to C$,
  - a next function $n : (a : A, R(c(a))) \to A$.

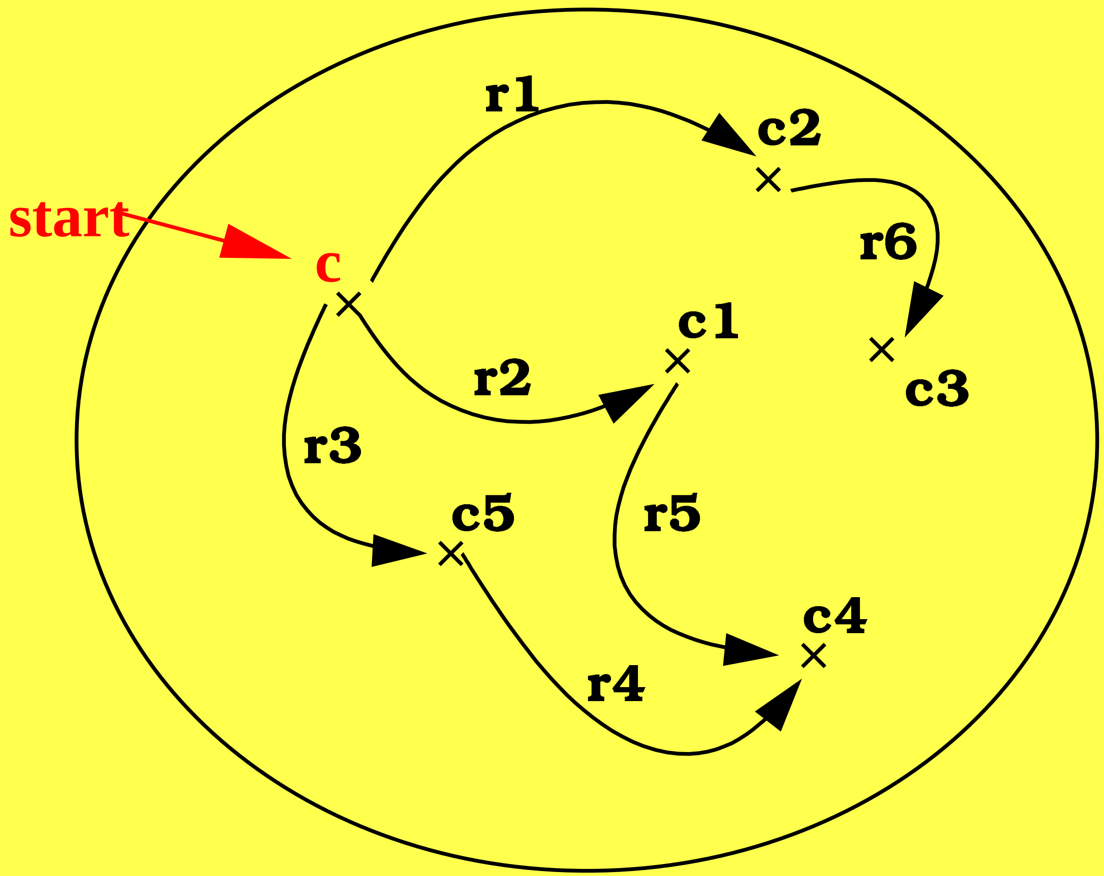
Graphs for F_0^∞

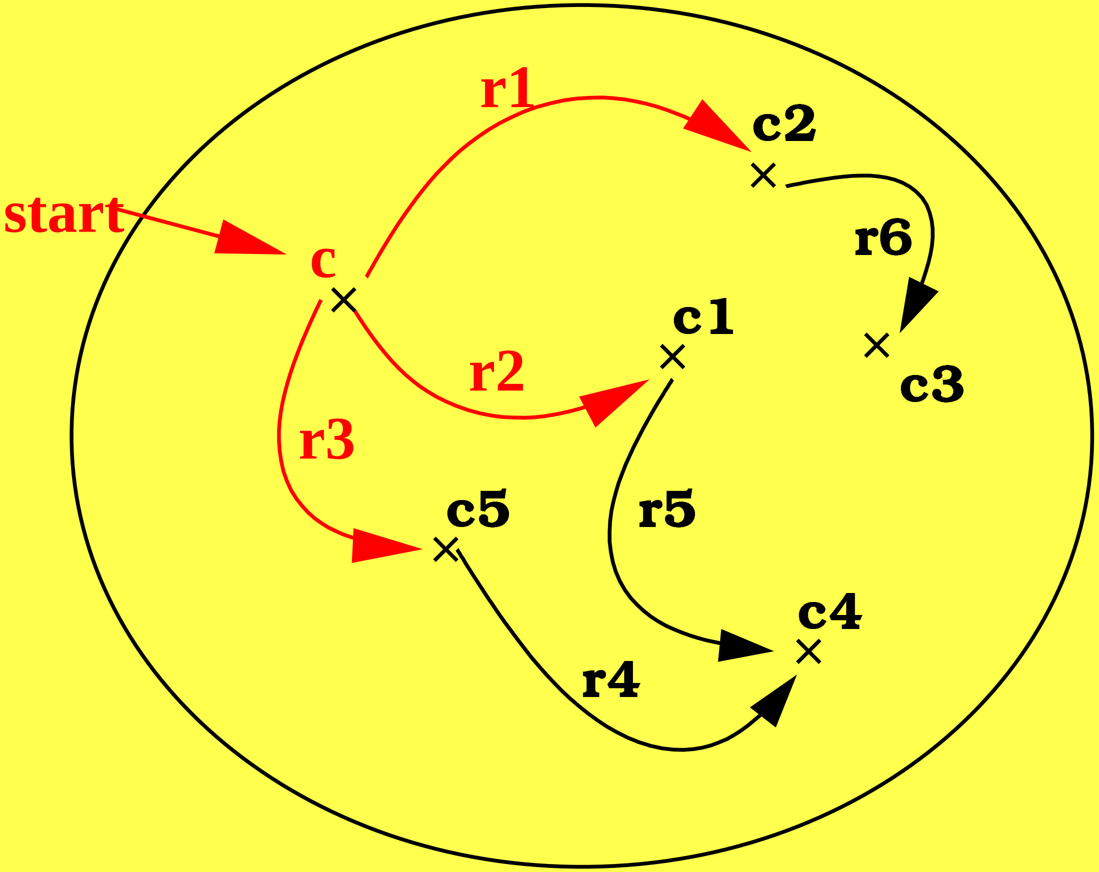
Assume $F(X) = \Sigma c : C.R(c) \to X$.

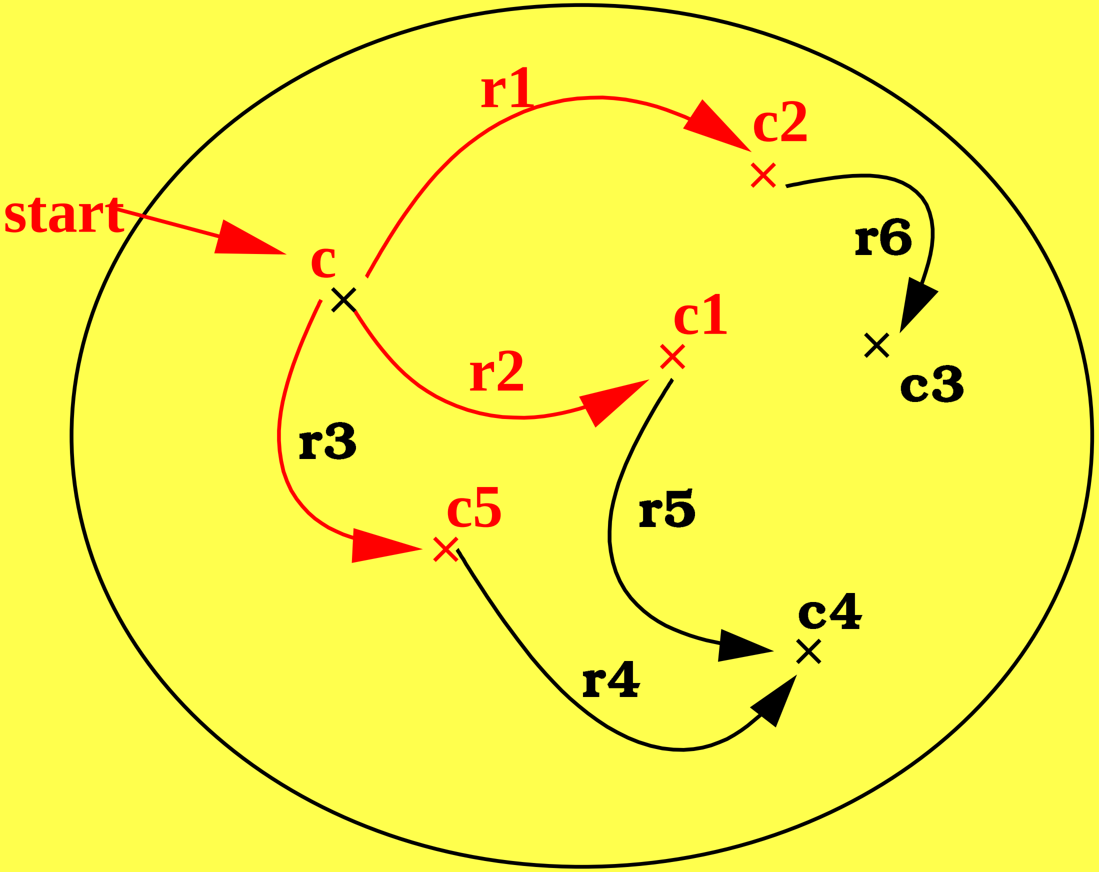
- A **graph** for F consists of
  - a set A,
  - a labelling function $c : A \to C$,
  - a next function $n : (a : A, R(c(a))) \to A$.
  - a starting node $a : A$.

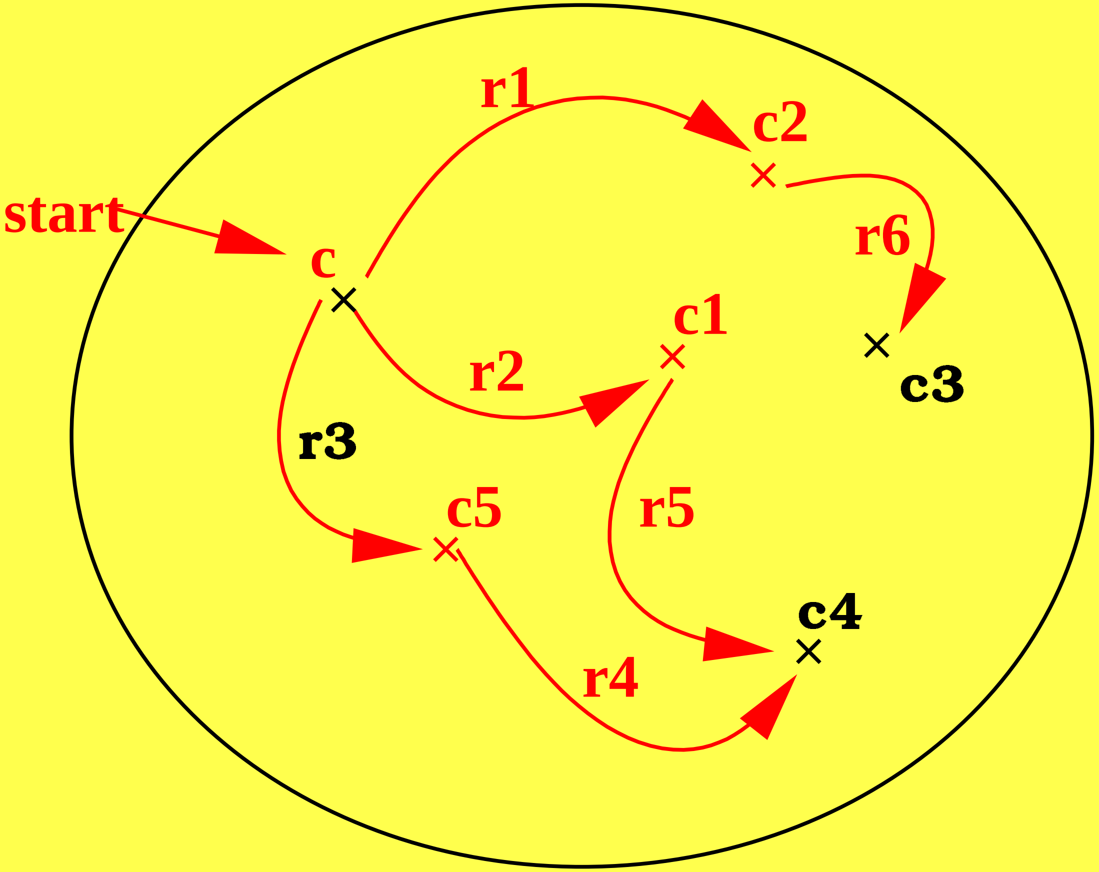


start
c
r1
c2
r6
c1
c3
r2
r3
c5
r5
c4
r4


start
c
r1
c2
r6
c1
r2
c3
r3
c5
r5
c4
r4


r1
c2
start
c
r6
c1
r2
c3
r3
c5
r5
c4
r4


r1
c2
start
r6
c
c1
r2
c3
r3
c5
r5
c4
r4


r1
c2
start
r6
c
c1
r2
c3
r3
c5
r5
c4
r4

More Abstractly

More Abstractly

- A graph for F consists of

More Abstractly

- A graph for F consists of
  - a set A,

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to (\Sigma c : C.R(c) \to A)$,

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to \underbrace{(\Sigma c : C.R(c) \to A)}_{F(A)}$,

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to \underbrace{(\Sigma c : C.R(c) \to A)}_{F(A)}$,
  - an $a : A$.

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to \underbrace{(\Sigma c : C.R(c) \to A)}_{F(A)}$,
  - an $a : A$.

- Introduction rule for F_0^∞:

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to \underbrace{(\Sigma c : C.R(c) \to A)}_{F(A)}$,
  - an $a : A$.

- Introduction rule for F_0^∞:
  every graph introduces an element of F_0^∞.

More Abstractly

- A graph for F consists of
  - a set A,
  - an $f : A \to \underbrace{(\Sigma c : C.R(c) \to A)}_{F(A)}$,
  - an $a : A$.

- Introduction rule for F_0^∞:
  every graph introduces an element of F_0^∞.

- However: no full elimination – Only: $\mathrm{elim} : F_0^\infty \to F(F_0^\infty)$.

Rules for Coiteration

Rules for Coiteration

Formation

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Introduction

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(A) \quad a : A}{\mathrm{intro}'(A, \gamma, a) : F_0^\infty}$$

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(A) \quad a : A}{\mathrm{intro}'(A, \gamma, a) : F_0^\infty}$$

Elimination

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(A) \quad a : A}{\mathrm{intro}'(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Rules for Coiteration

Formation

$$F_0^\infty : \mathrm{Set}$$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(A) \quad a : A}{\mathrm{intro}'(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Equality

Rules for Coiteration

Formation

$$F_0^{\infty} : \mathrm{Set}$$

Introduction

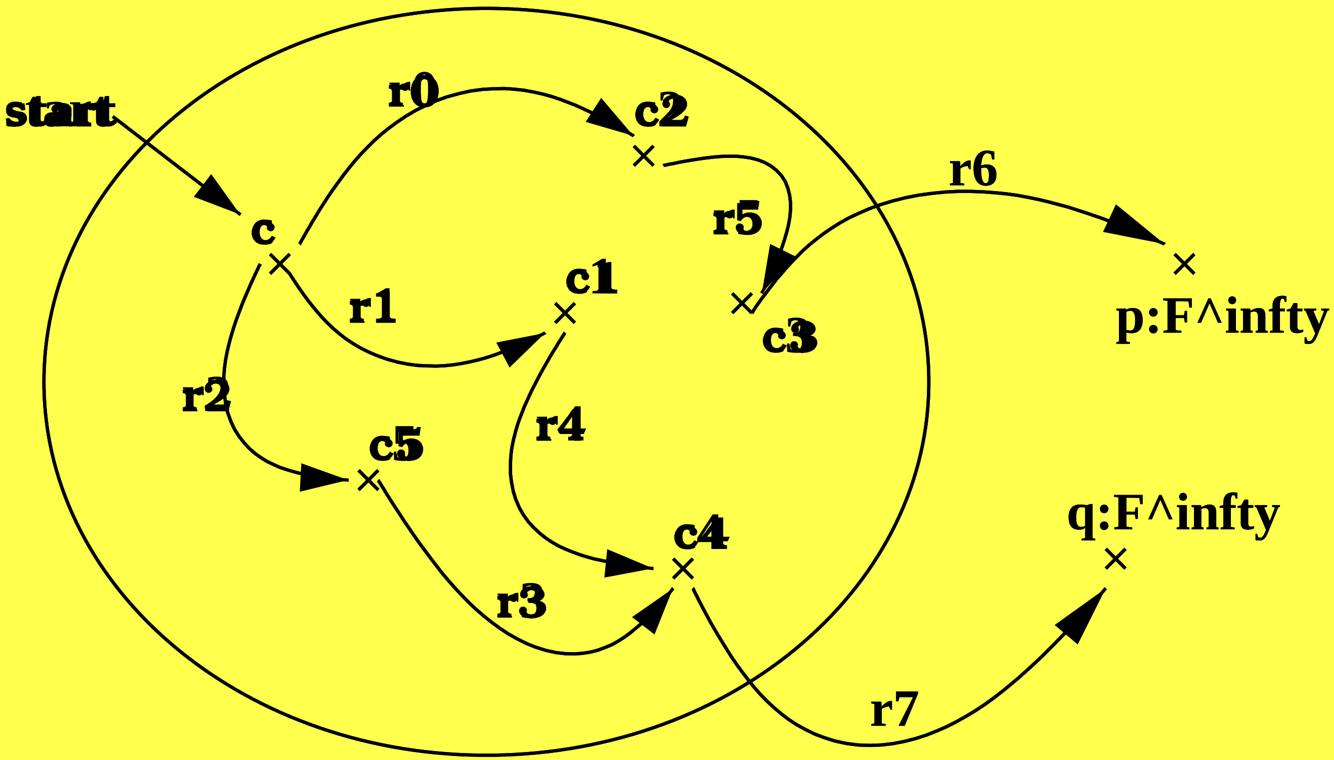
$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(A) \quad a : A}{\mathrm{intro}'(A, \gamma, a) : F_0^{\infty}}$$

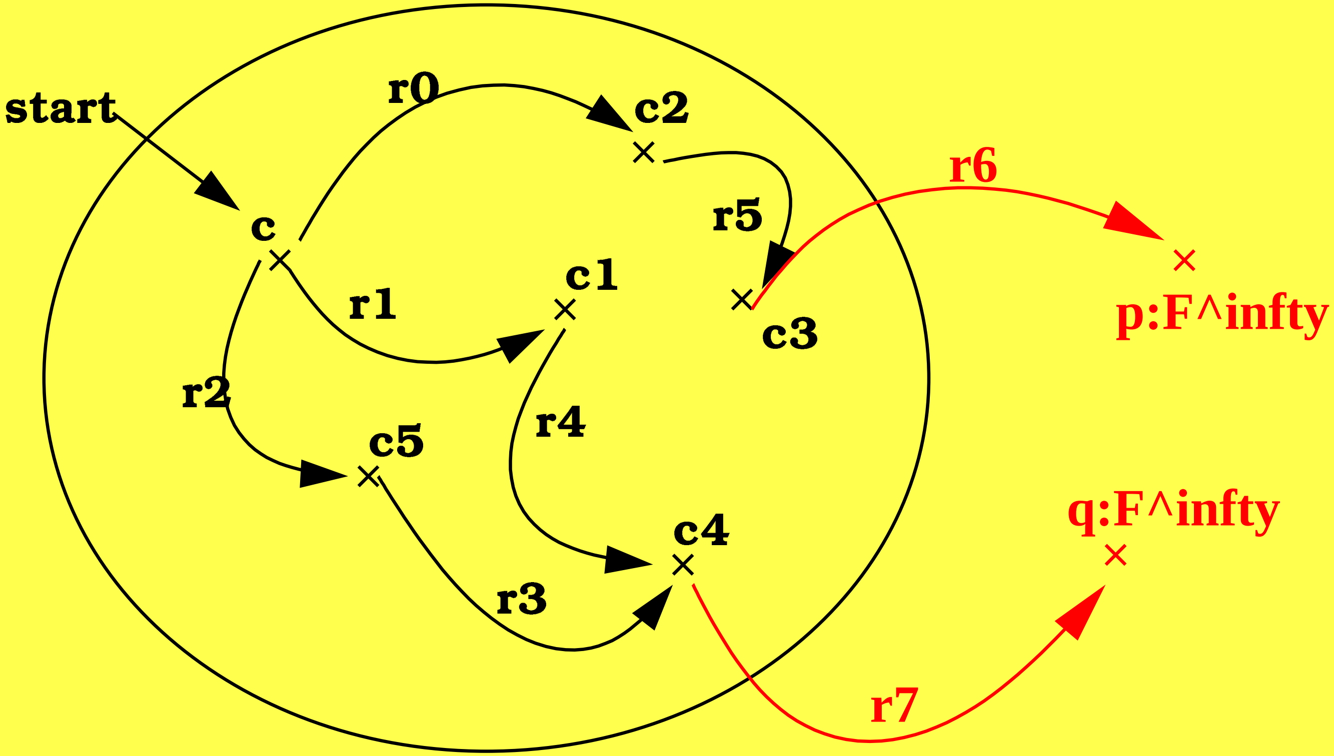
Elimination

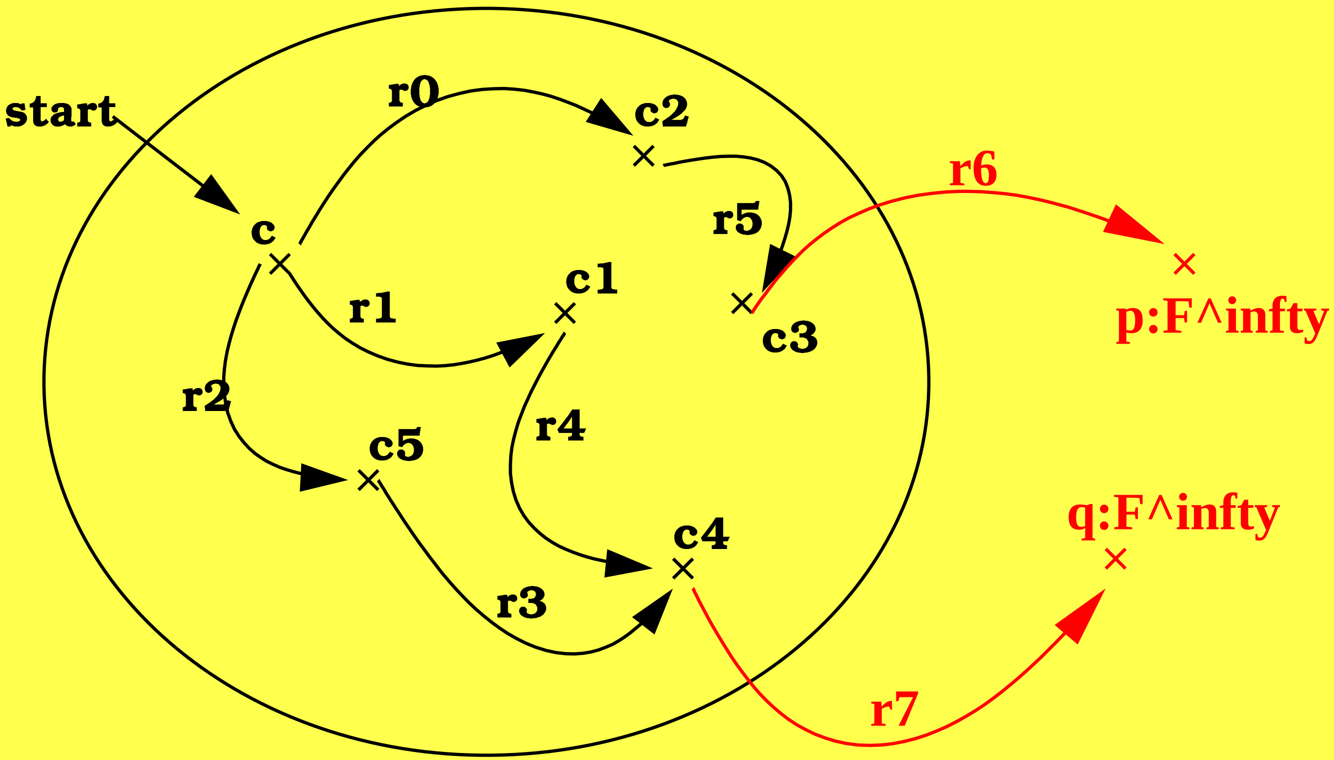
$$\frac{p : F_0^{\infty}}{\mathrm{elim}(p) : F(F_0^{\infty})}$$

Equality

$$\begin{array}{rcl} \mathrm{elim}(\mathrm{intro}'(A, \gamma, a)) & = & F(\lambda x.\mathrm{intro}'(A, \gamma, x))(\gamma(a)) \\ & : & F(F_0^{\infty}) \end{array}$$


start
r0
c2
c
r5
r6
r1
c1
c3
p:F^infty
r2
r4
c5
c4
q:F^infty
r3
r7


start
c
r0
c2
r1
c1
r5
c3
r6
p:F^infty
r2
c5
r4
c4
r3
q:F^infty
r7



Easier to define successor —— Definable using Coiteration

Rules for Corecursion

Rules for Corecursion

Formation

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Elimination

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Equality

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Equality

$$\mathrm{elim}(\mathrm{intro}(A, \gamma, a)) \ = F(f)(\gamma(a)) : F(F_0^\infty)$$

Rules for Corecursion

Formation $F_0^\infty : \mathrm{Set}$

Introduction

$$\frac{A : \mathrm{Set} \quad \gamma : A \to F(\mathrm{cont}(A) + \mathrm{fin}(\mathbf{F_0^\infty})) \quad a : A}{\mathrm{intro}(A, \gamma, a) : F_0^\infty}$$

Elimination

$$\frac{p : F_0^\infty}{\mathrm{elim}(p) : F(F_0^\infty)}$$

Equality

$$\mathrm{elim}(\mathrm{intro}(A, \gamma, a)) = F(f)(\gamma(a)) : F(F_0^\infty)$$

where $f(\mathrm{cont}(a)) = \mathrm{intro}(A, \gamma, a)$

$f(\mathrm{fin}(p)) = p$

$$\boxed{\mathbf{F}^{\infty}(A)}$$

$\mathbf{F}^{\infty}(A)$

- Want to construct a functor based on F_0^{∞}.

$\boxed{\mathbf{F}^\infty(A)}$

- Want to construct a functor based on F_0^∞.
- Idea: start from atomic elements $(a : A)$ and "build possibly non-well-founded many construtors of F on top of it".

$\boxed{\mathbf{F}^{\infty}(A)}$

- Want to construct a functor based on F_0^{∞}.

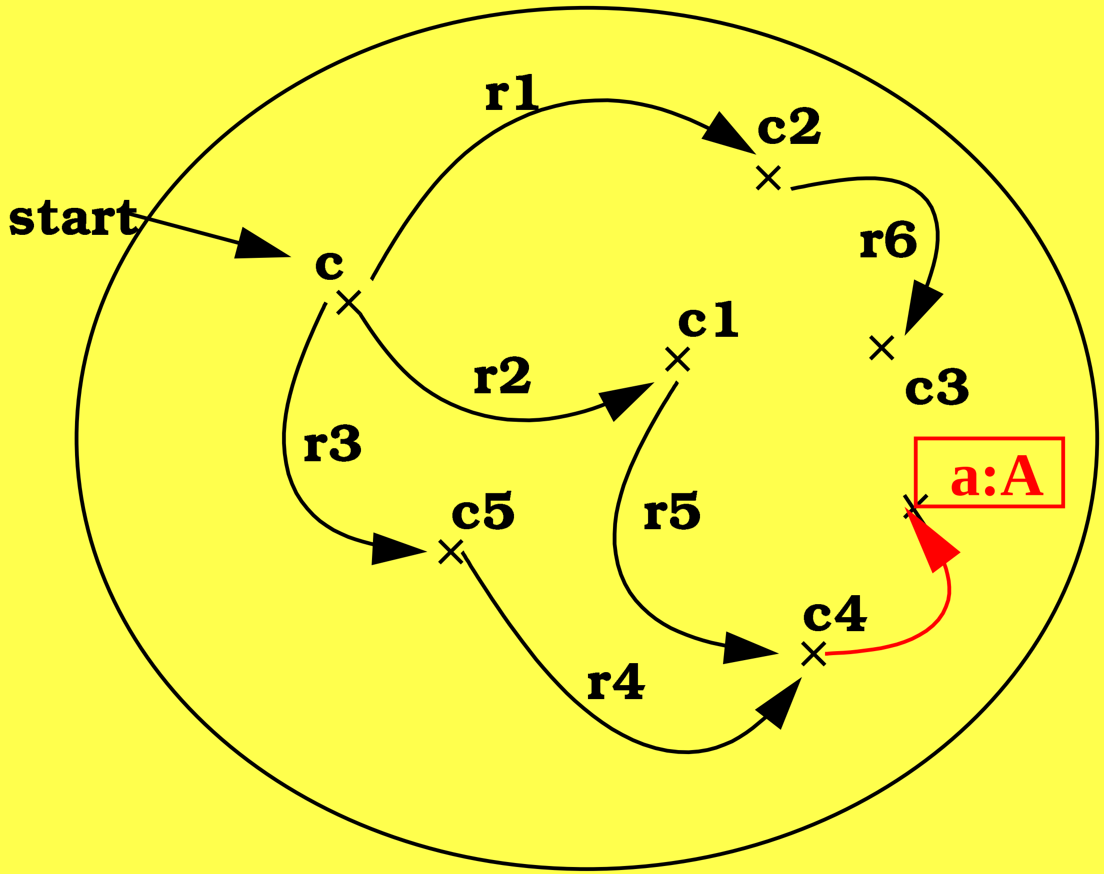
- Idea: start from atomic elements $(a : A)$ and "build possibly non-well-founded many construtors of F on top of it".

- More precisely: Let $F_A := \lambda X.\mathrm{at}(A) + \mathrm{do}(F(X))$.

$\mathbf{F}^{\infty}(A)$

- Want to construct a functor based on F_0^{∞}.
- Idea: start from atomic elements $(a : A)$ and "build possibly non-well-founded many construtors of F on top of it".
- More precisely: Let $F_A := \lambda X.\mathrm{at}(A) + \mathrm{do}(F(X))$.
- $\mathbf{F}^{\infty}(\mathbf{A}) := (F_A)_0^{\infty}$.

Graphs for $\mathbf{F}^{\infty}(\mathbf{A})$



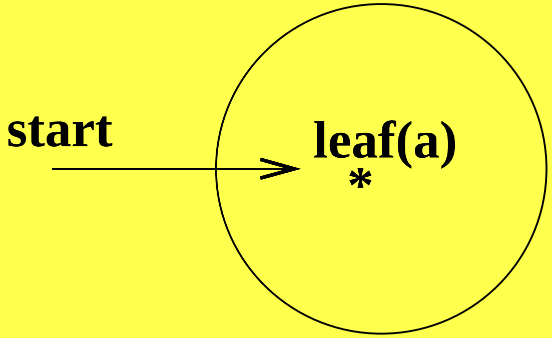
3. Some Operations on Coalgebras

3. Some Operations on Coalgebras

Atoms

3. Some Operations on Coalgebras

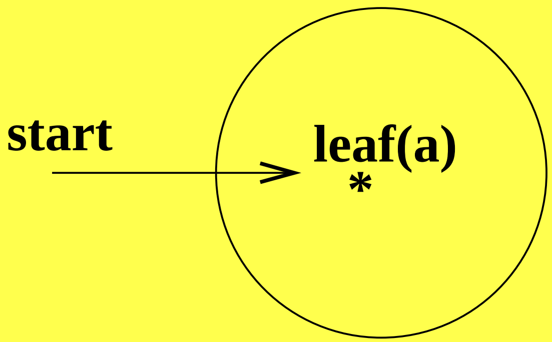
Atoms



3. Some Operations on Coalgebras

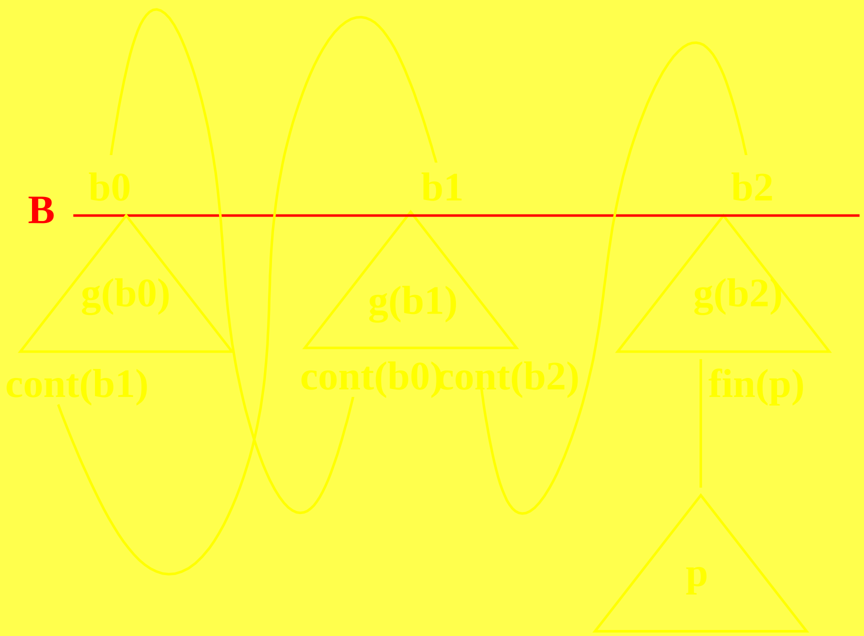
Atoms

For $a : A$ let $\underset{\sim\sim\sim\sim\sim}{\mathrm{At}(a)} := \mathrm{intro}(\{\star\}, \lambda x.\mathrm{cont}(\mathrm{at}(a)), \star) : F^{\infty}(A)$.



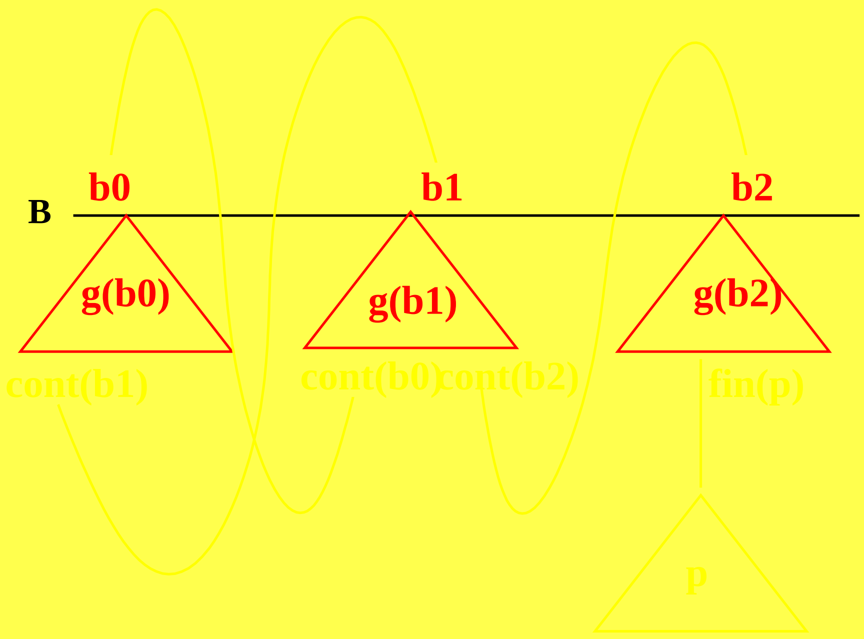
Repeat

Repeat



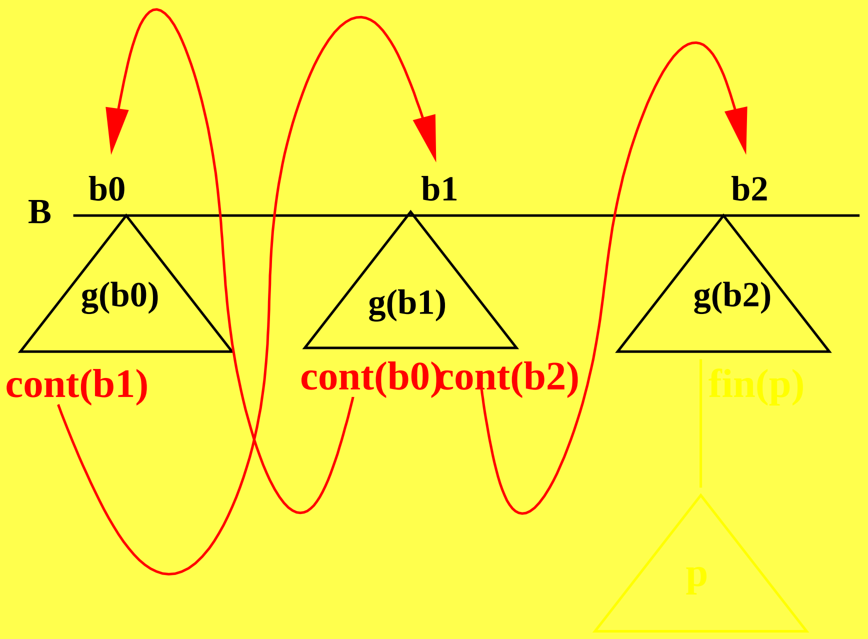
$B : \mathrm{Set}$

Repeat



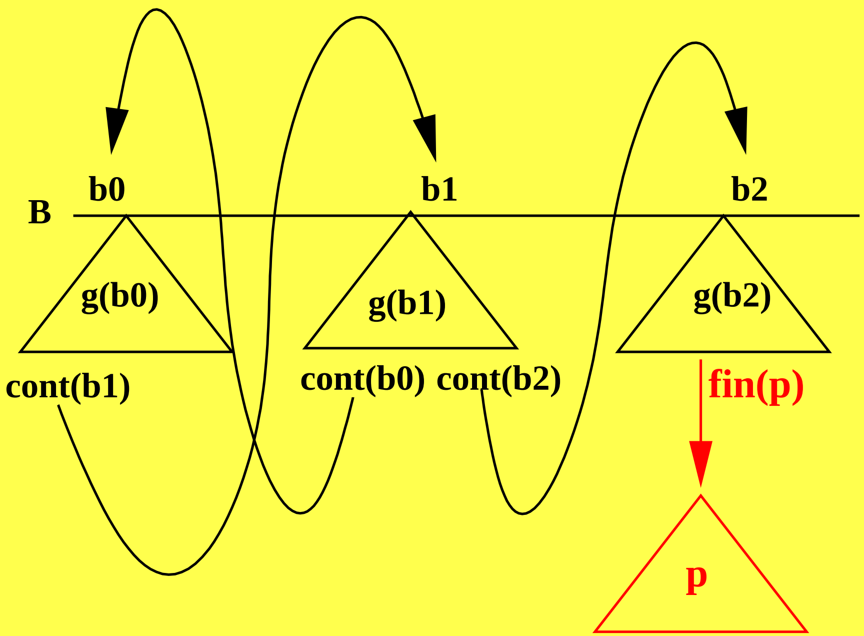
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



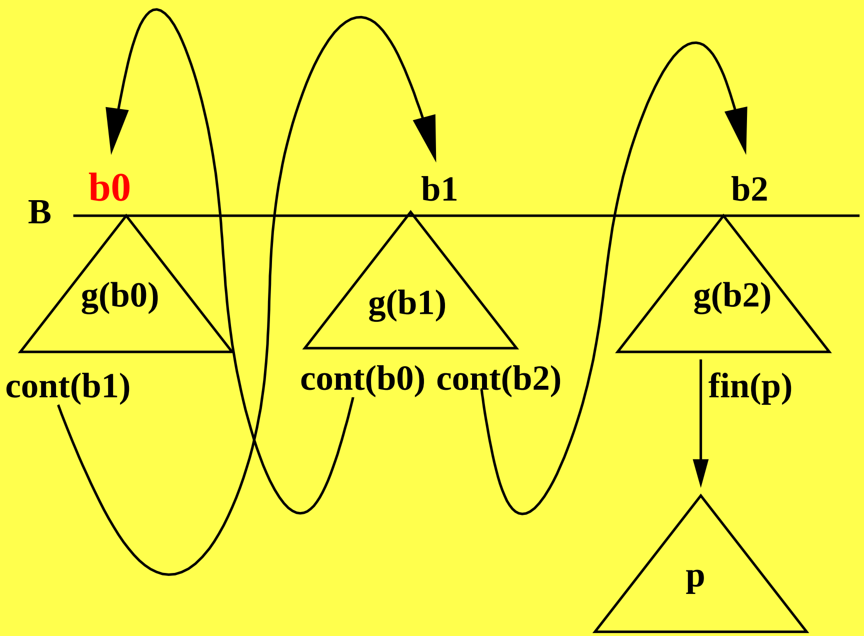
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



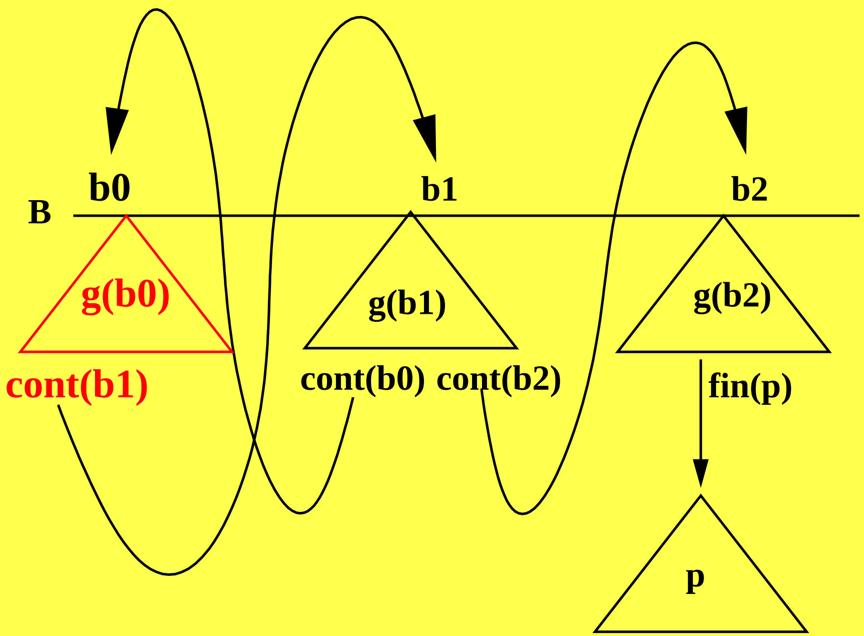
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



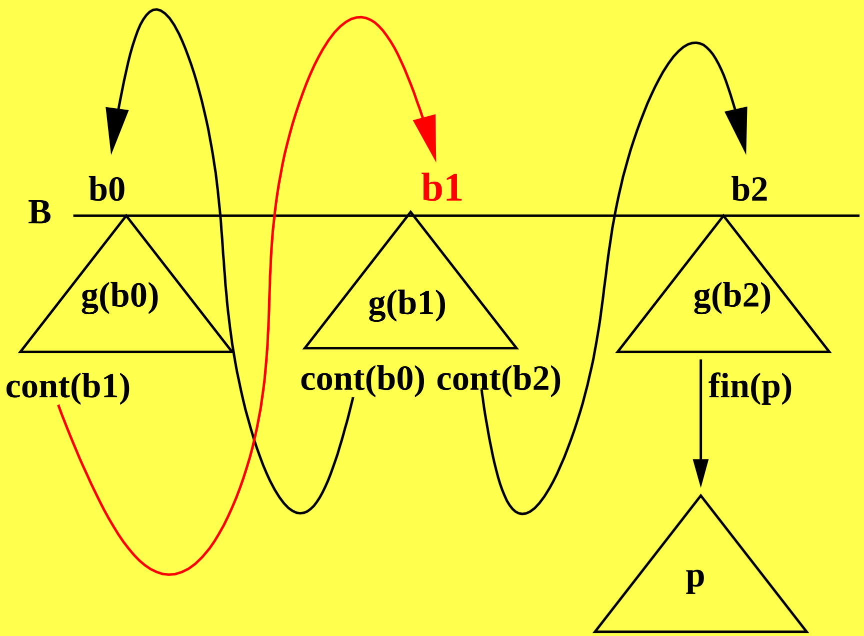
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



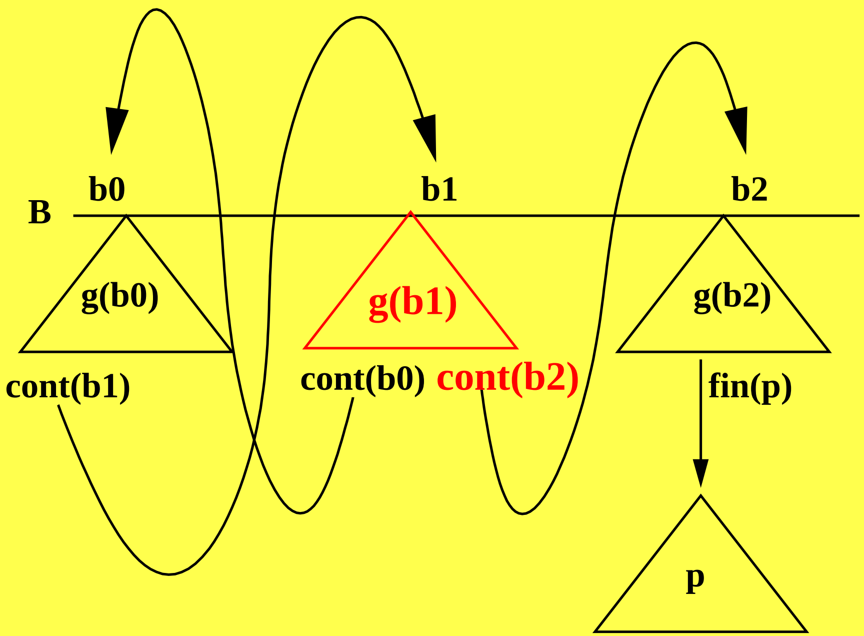
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



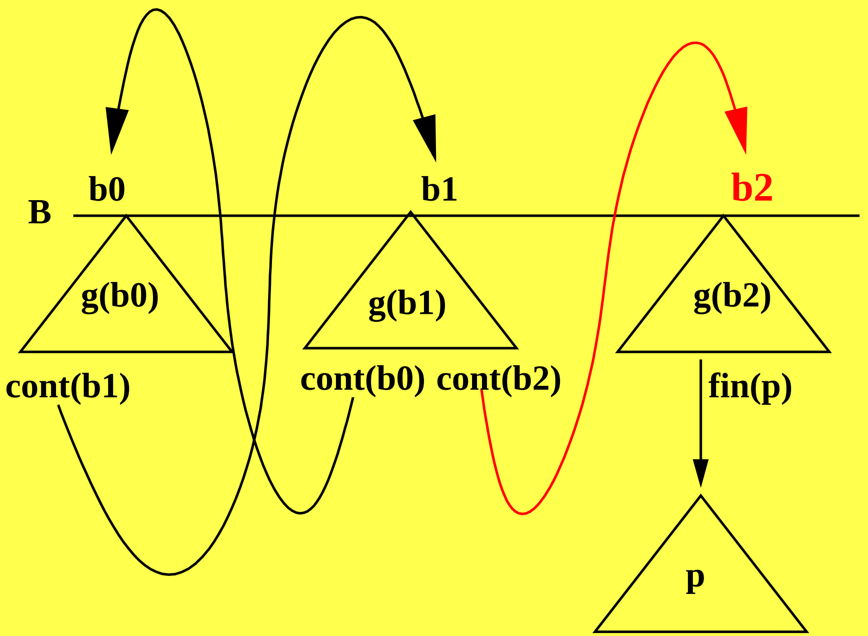
$$B : \mathrm{Set} \qquad g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$$

Repeat



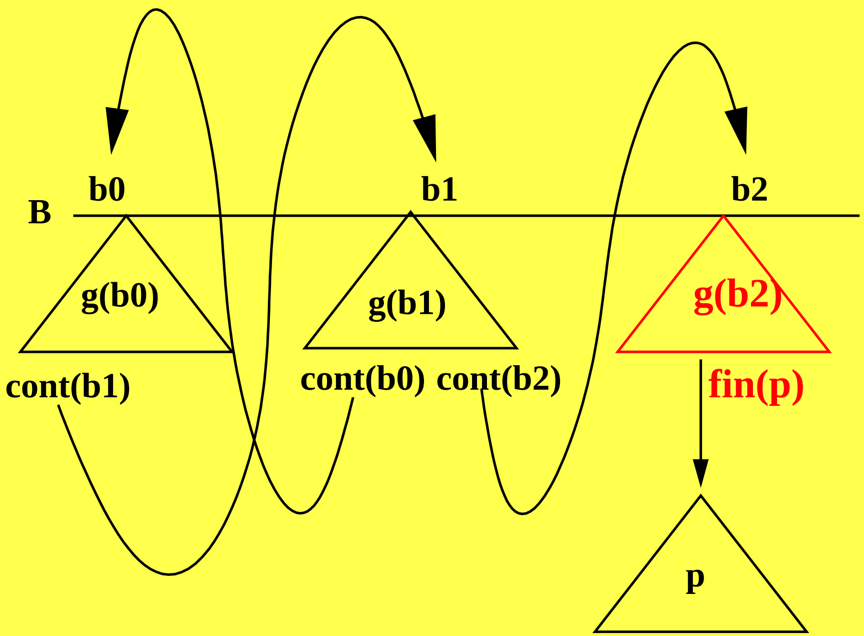
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



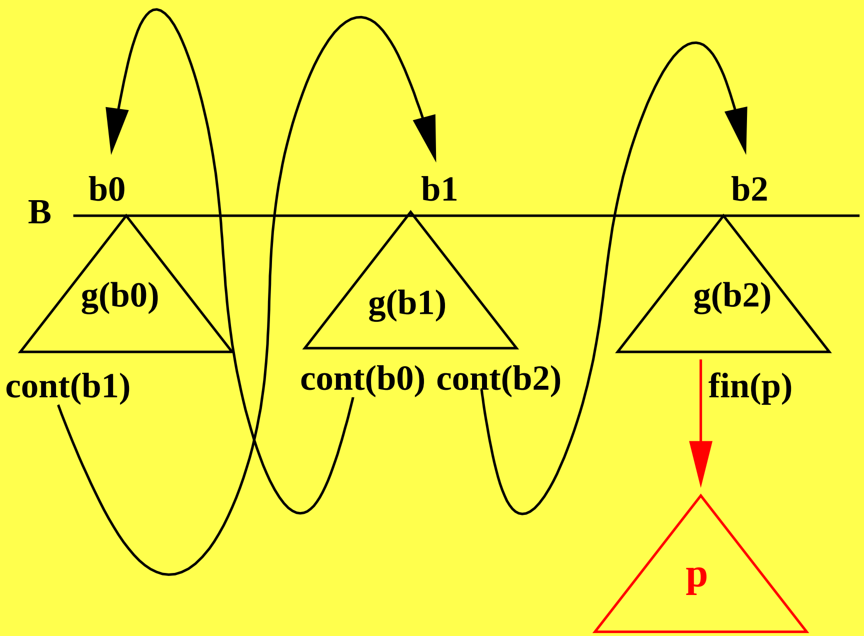
$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

Repeat



$$B : \mathrm{Set} \qquad g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$$

$\mathbf{repeat(B, g, b)}$

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$, $b : B$.

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$, $b : B$.

- Define $\underset{\sim}{\mathrm{repeat}(B, g, b)} := \mathrm{intro}(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$, $b : B$.

- Define $\underset{\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim\sim}{\mathrm{repeat}(B, g, b)} := \mathrm{intro}(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

- where we define

$$f : F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})) \to (\mathrm{cont}(F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))) + \mathrm{fin}(F_0^{\infty}))$$

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$, $b : B$.

- Define $\mathrm{repeat}(B, g, b) := \mathrm{intro}(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})), f, \mathrm{At}(\mathrm{cont}(b)))$

- where we define

$$f : F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})) \to (\mathrm{cont}(F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))) + \mathrm{fin}(F_0^{\infty}))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$,

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))$, $b : B$.

- Define $\mathrm{repeat}(B, g, b) := \mathrm{intro}(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

- where we define

$$f : F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})) \to (\mathrm{cont}(F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F_0^{\infty})))) + \mathrm{fin}(F_0^{\infty}))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$, then $f(a) = \mathrm{cont}(\underbrace{g(b)}_{:F(F^{\infty}(\mathrm{fin}(F_0^{\infty})+\mathrm{cont}(B)))})$.

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$, $b : B$.

- Define $\underset{\sim\sim\sim}{\mathrm{repeat}(B, g, b)} := \mathrm{intro}(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)), f, \mathrm{At}(\mathrm{cont}(b))\ )$

- where we define

$$f : F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)) \to (\mathrm{cont}(F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))) + \mathrm{fin}(F_0^\infty))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$, then $f(a) = \mathrm{cont}(\underbrace{g(b)}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)))})$.
  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{fin}(p))$,

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$, $b : B$.

- Define $\underset{\sim\sim\sim\sim\sim\sim\sim}{\mathrm{repeat}(B, g, b)} := \mathrm{intro}(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

- where we define

$$f : F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)) \to (\mathrm{cont}(F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))) + \mathrm{fin}(F_0^\infty))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$, then $f(a) = \mathrm{cont}(\underbrace{g(b)}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)))})$.
  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{fin}(p))$, then $f(a) = \mathrm{fin}(p)$.

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$, $b : B$.

- Define $\mathrm{repeat}(B, g, b) := \mathrm{intro}(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

- where we define

$$f : F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)) \longrightarrow (\mathrm{cont}(F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))) + \mathrm{fin}(F_0^\infty))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$, then $f(a) = \mathrm{cont}(\underbrace{g(b)}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)))})$.
  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{fin}(p))$, then $f(a) = \mathrm{fin}(p)$.
  - if $\mathrm{elim}(a) = \mathrm{do}(\underbrace{p}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)a))})$,

$\mathbf{repeat(B, g, b)}$

- Assume $B : \mathrm{Set}$, $g : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))$, $b : B$.

- Define $\underset{\sim}{\mathrm{repeat}(B, g, b)} := \mathrm{intro}(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)), f, \mathrm{At}(\mathrm{cont}(b))\ \ )$

- where we define

$$f : F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)) \longrightarrow (\mathrm{cont}(F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F_0^\infty)))) + \mathrm{fin}(F_0^\infty))$$

  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{cont}(b))$, then $f(a) = \mathrm{cont}(\underbrace{g(b)}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)))})$.
  - if $\mathrm{elim}(a) = \mathrm{at}(\mathrm{fin}(p))$, then $f(a) = \mathrm{fin}(p)$.
  - if $\mathrm{elim}(a) = \mathrm{do}(\underbrace{p}_{:F(F^\infty(\mathrm{fin}(F_0^\infty)+\mathrm{cont}(B)a))})$, then $f(a) = \mathrm{cont}(p)$.

4. The μ-Operator and Coalgebras

4. The μ-Operator and Coalgebras

- $\mathrm{NStream} = F_0^\infty$, where
  $F(X) = \mathrm{data}\{\mathrm{cons}(n : \mathrm{N}, x : X)\} \quad (\approx \mathrm{N} \times X)$

4. The μ-Operator and Coalgebras

- $\mathrm{NStream} = F_0^\infty$, where
  $F(X) = \mathrm{data}\{\mathrm{cons}(n : \mathrm{N}, x : X)\} \quad (\approx \mathrm{N} \times X)$

- Elements of $\mathrm{NStream}$ have the form
  $\mathrm{cons}(n_1, \mathrm{cons}(n_2, \mathrm{cons}(n_3, \cdots)))$.

4. The μ-Operator and Coalgebras

- $\mathrm{NStream} = F_0^\infty$, where
  $F(X) = \mathrm{data}\{\mathrm{cons}(n : \mathrm{N}, x : X)\} \quad (\approx \mathrm{N} \times X)$

- Elements of $\mathrm{NStream}$ have the form
  $\mathrm{cons}(n_1, \mathrm{cons}(n_2, \mathrm{cons}(n_3, \cdots)))$.

- Would like to define elements of $\mathrm{NStream}$ recursively.

4. The μ-Operator and Coalgebras

- $\mathrm{NStream} = F_0^\infty$, where
  $F(X) = \mathrm{data}\{\mathrm{cons}(n : \mathrm{N}, x : X)\} \quad (\approx \mathrm{N} \times X)$

- Elements of $\mathrm{NStream}$ have the form
  $\mathrm{cons}(n_1, \mathrm{cons}(n_2, \mathrm{cons}(n_3, \cdots)))$.

- Would like to define elements of $\mathrm{NStream}$ recursively. E.g.
  - $f : \mathrm{N} \to \mathrm{NStream}$, $f(n) = \mathrm{cons}(n, f(n+1))$.

4. The μ-Operator and Coalgebras

- $\mathrm{NStream} = F_0^\infty$, where
  $F(X) = \mathrm{data}\{\mathrm{cons}(n : \mathrm{N}, x : X)\} \quad (\approx \mathrm{N} \times X)$

- Elements of $\mathrm{NStream}$ have the form
  $\mathrm{cons}(n_1, \mathrm{cons}(n_2, \mathrm{cons}(n_3, \cdots)))$.

- Would like to define elements of $\mathrm{NStream}$ recursively. E.g.
  - $f : \mathrm{N} \to \mathrm{NStream}$, $f(n) = \mathrm{cons}(n, f(n+1))$.

- In this form non-normalizing.

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

- f as above can be defined as $\mu_{\mathrm{N}}(\lambda g, n.\mathrm{cons}(n, g(n+1)))$.

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

- f as above can be defined as $\mu_{\mathrm{N}}(\lambda g, n.\mathrm{cons}(n, g(n+1)))$.

- μ is a constructor $\Rightarrow$ recursion evaluated only when applying elim.

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

- f as above can be defined as $\mu_{\mathrm{N}}(\lambda g, n.\mathrm{cons}(n, g(n+1)))$.

- μ is a constructor $\Rightarrow$ recursion evaluated only when applying elim.

- In order to define elim, we need to apply elim to the body of μ.

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

- f as above can be defined as $\mu_{\mathrm{N}}(\lambda g, n.\mathrm{cons}(n, g(n+1)))$.

- μ is a constructor $\Rightarrow$ recursion evaluated only when applying elim.

- In order to define elim, we need to apply elim to the body of μ.
  Better: replace the type of g above by:

  $$g : (A \to \mathrm{NStream}) \to A \to F(\mathrm{NStream}) \enspace .$$

The μ-Operator

- Instead try a constructor μ. (Idea from T. Coquand).
  Assume $A : \mathrm{Set}$, $g : (A \to \mathrm{NStream}) \to A \to \mathrm{NStream}$.
  Then $\mu_A(g) : A \to \mathrm{NStream}$.

- f as above can be defined as $\mu_{\mathrm{N}}(\lambda g, n.\mathrm{cons}(n, g(n+1)))$.

- μ is a constructor $\Rightarrow$ recursion evaluated only when applying elim.

- In order to define elim, we need to apply elim to the body of μ.
  Better: replace the type of g above by:

  $$g : (A \to \mathrm{NStream}) \to A \to F(\mathrm{NStream}) \enspace .$$

- Now define $\mathrm{elim}(\mu_A(g, a)) = g(\mu_A(g), a)$.

Problems with the μ-Operator

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.
    Not normalizing.

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.
    Not normalizing. Instead demand
    * In $\mu_A(\lambda g, x.t)$, elim should not be applied to a term depending on g.

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.
    Not normalizing. Instead demand
    - In $\mu_A(\lambda g, x.t)$, elim should not be applied to a term depending on g.

- Thierry Coquand calls such t **guarded**.
  (He demands as well one constructor to the outside.
  Automatically fulfilled because of the type of t).

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.
    Not normalizing. Instead demand
    - In $\mu_A(\lambda g, x.t)$, elim should not be applied to a term depending on g.

- Thierry Coquand calls such t **guarded**.
  (He demands as well one constructor to the outside.
  Automatically fulfilled because of the type of t).

- **Principle of guarded induction**:
  Elements of $F^\infty(A)$ are introduced by μ applied to guarded μ-terms.

Problems with the μ-Operator

- Example: $f = \mu_{\{*\}}(\lambda g, x.\mathrm{elim}(g(x)))$.
  - $\mathrm{elim}(f(x)) = (\lambda g, x.\mathrm{elim}(g(x)))(f, x) = \mathrm{elim}(f(x))$.
    Not normalizing. Instead demand
    - In $\mu_A(\lambda g, x.t)$, elim should not be applied to a term depending on g.

- Thierry Coquand calls such t **guarded**.
  (He demands as well one constructor to the outside.
  Automatically fulfilled because of the type of t).

- **Principle of guarded induction**:
  Elements of $F^\infty(A)$ are introduced by μ applied to guarded μ-terms.

- μ generalizes to arbitary F^∞.

Generating μ-Terms

Generating μ-Terms

- We had:

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F^{\infty}(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^{\infty}(A)$.

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \rightarrow F(F^{\infty}(\mathrm{cont}(B) + \mathrm{fin}(F^{\infty}(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^{\infty}(A)$.

- Define for f above

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^\infty(A)$.

- Define for f above

$$\widetilde{f} \quad : \quad (B \to F^\infty(A)) \to B \to F(F^\infty(A)) \ ,$$

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^\infty(A)$.

- Define for f above

$$\begin{array}{rcl} \widetilde{f} & : & (B \to F^\infty(A)) \to B \to F(F^\infty(A)) \enspace , \\ \widetilde{f}(g, b) & = & F(h_0)(F(F^\infty(h))(f(b))) \enspace , \end{array}$$

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^\infty(A)$.

- Define for f above

$$
\begin{array}{rcl}
\widetilde{f} & : & (B \to F^\infty(A)) \to B \to F(F^\infty(A)) \ , \\
\widetilde{f}(g, b) & = & F(h_0)(F(F^\infty(h))(f(b))) \ , \\
\text{where} & & \begin{array}{rcl} h & : & (\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))) \to F^\infty(A) \ , \\ h(\mathrm{cont}(b)) & = & g(b) \ , \\ h(\mathrm{fin}(p)) & = & p \ . \end{array}
\end{array}
$$

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^\infty(A)$.

- Define for f above

$$
\begin{array}{rcl}
\widetilde{f} & : & (B \to F^\infty(A)) \to B \to F(F^\infty(A)) \ , \\
\widetilde{f}(g, b) & = & F(h_0)(F(F^\infty(h))(f(b))) \ , \\
\text{where} & & \begin{array}{rcl} h & : & (\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))) \to F^\infty(A) \ , \\ h(\mathrm{cont}(b)) & = & g(b) \ , \\ h(\mathrm{fin}(p)) & = & p \ . \end{array} \\
\text{and} & & h_0 : F^\infty(F^\infty(A)) \to F^\infty(A) \ .
\end{array}
$$

Generating μ-Terms

- We had:
  - If $B : \mathrm{Set}$, $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$, $b : B$, then $\mathrm{repeat}(B, f, b) : F^\infty(A)$.

- Define for f above

$$\begin{array}{rcl}
\widetilde{f} & : & (B \to F^\infty(A)) \to B \to F(F^\infty(A)) \enspace , \\
\widetilde{f}(g, b) & = & F(h_0)(F(F^\infty(h))(f(b))) \enspace , \\
\text{where} & & \begin{array}{rcl} h & : & (\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))) \to F^\infty(A) \enspace , \\ h(\mathrm{cont}(b)) & = & g(b) \enspace , \\ h(\mathrm{fin}(p)) & = & p \enspace . \end{array} \\
\text{and} & & h_0 : F^\infty(F^\infty(A)) \to F^\infty(A) \enspace .
\end{array}$$

- Then $\mathrm{repeat}(B, f, b) = \mu_B(\widetilde{f}, b)$.

Comparison of μ and repeat

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.
  - But now infinitely many constructors of F (even unbounded chains) can be applied to it.

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.
  - But now infinitely many constructors of F (even unbounded chains) can be applied to it.
  - No longer syntactic condition.

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.
  - But now infinitely many constructors of F (even unbounded chains) can be applied to it.
  - No longer syntactic condition.
  - Subsumes all cases of functions definable by guarded induction principle, but extends this notion.

Comparison of μ and repeat

- Consider for $f : B \to F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.
  - But now infinitely many constructors of F (even unbounded chains) can be applied to it.
  - No longer syntactic condition.
  - Subsumes all cases of functions definable by guarded induction principle, but extends this notion.

- If we replace type of f by $F(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A)))$
  then $\widetilde{f}$ can be defined by the guarded induction principle.

Comparison of μ and repeat

- Consider for $f : B \rightarrow F(F^\infty(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A))))$
  $\widetilde{f} := \lambda g, b.F(h_0)(F(F^\infty(h))(f(b)))$.

- Now $\widetilde{f}$ is "extended guarded": no elim applied to a term containing g.
  - But now infinitely many constructors of F (even unbounded chains) can be applied to it.
  - No longer syntactic condition.
  - Subsumes all cases of functions definable by guarded induction principle, but extends this notion.

- If we replace type of f by $F(\mathrm{cont}(B) + \mathrm{fin}(F^\infty(A)))$
  then $\widetilde{f}$ can be defined by the guarded induction principle.
  - Suffices (together with At) to define intro.

- Therefore functions definable by guarded induction principle and by our rules are the same.

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules $=$ implemented ones. (Constructor).	

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules $=$ implemented ones. (Constructor).	**Elimination Rules** Formal rules $=$ implemented ones. (elim or case distinction)

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules = implemented ones. (Constructor).	**Elimination Rules** Formal rules = implemented ones. (elim or case distinction)
Elimination Rules Formal rules = recursion operator.	

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules = implemented ones. (Constructor).	**Elimination Rules** Formal rules = implemented ones. (elim or case distinction)
Elimination Rules Formal rules = recursion operator. Implemented ones= pattern matching +termination check	

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules = implemented ones. (Constructor).	**Elimination Rules** Formal rules = implemented ones. (elim or case distinction)
Elimination Rules Formal rules = recursion operator. Implemented ones= pattern matching +termination check Syntactic condition.	

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules = implemented ones. (Constructor).	**Elimination Rules** Formal rules = implemented ones. (elim or case distinction)
Elimination Rules Formal rules = recursion operator. Implemented ones= pattern matching +termination check Syntactic condition.	**Introduction Rules** Formal rules =intro.

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules $=$ implemented ones. (Constructor).	**Elimination Rules** Formal rules $=$ implemented ones. (elim or case distinction)
Elimination Rules Formal rules $=$ recursion operator. Implemented ones$=$ pattern matching $+$termination check Syntactic condition.	**Introduction Rules** Formal rules $=$intro. Implemented ones$=$ μ $+$ guardedness check $+\mathrm{At}$ (Atom)

- Therefore functions definable by guarded induction principle and by our rules are the same.

Formal Calculus and Implementations

Algebras	Coalgebras
Introduction Rules Formal rules $=$ implemented ones. (Constructor).	**Elimination Rules** Formal rules $=$ implemented ones. (elim or case distinction)
Elimination Rules Formal rules $=$ recursion operator. Implemented ones$=$ pattern matching $+$termination check Syntactic condition.	**Introduction Rules** Formal rules $=$intro. Implemented ones$=$ μ $+$ guardedness check $+\mathrm{At}$ (Atom) Syntactic condition.

Conclusion

Conclusion

- Rules for coiteration seem to be the appropriate ones.

Conclusion

- Rules for coiteration seem to be the appropriate ones.
- μ-operator $=$ correct principle for implementations.

Conclusion

- Rules for coiteration seem to be the appropriate ones.
- μ-operator $=$ correct principle for implementations.
- Both allow to define non-terminating programs in a hopefully normalizing type theory.

Conclusion

- Rules for coiteration seem to be the appropriate ones.
- μ-operator $=$ correct principle for implementations.
- Both allow to define non-terminating programs in a hopefully normalizing type theory.
- Model can be defined.

Conclusion

- Rules for coiteration seem to be the appropriate ones.
- μ-operator $=$ correct principle for implementations.
- Both allow to define non-terminating programs in a hopefully normalizing type theory.
- Model can be defined.
- Normalization still to be shown.

Conclusion

- Rules for coiteration seem to be the appropriate ones.
- μ-operator $=$ correct principle for implementations.
- Both allow to define non-terminating programs in a hopefully normalizing type theory.
- Model can be defined.
- Normalization still to be shown.
- Extension to dependent coalgebras exists.
  Dependent introduction rule for (dependent) coalgebras
  $=$ analogue of dependent elimination rule for algebras.