A Concrete Introduction to Abstract Coinductive Datatypes

Andrei Popescu

Middlesex University
School of Science and Technology
Foundations of Computing Group

This is continuation of

`www.andreipopescu.uk/resourcesForStudents/introductionToDatatypes.pdf`

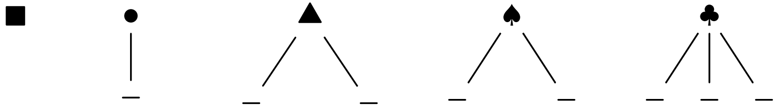
See also

`www.andreipopescu.uk/resourcesForStudents/codatatypesInIsabelleHOL.pdf`

`www.andreipopescu.uk/slides/ESOP2015-slides.pdf`

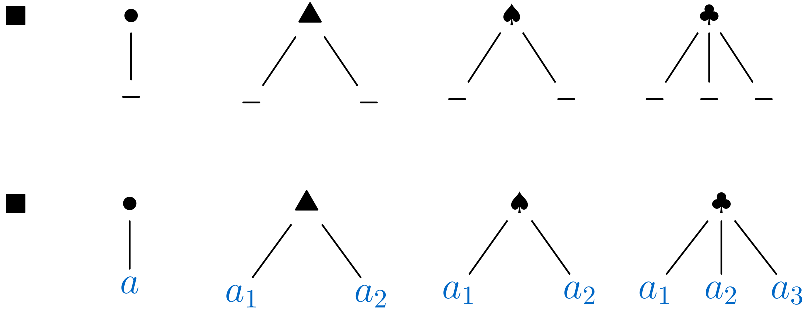
Recall: It's All About Shape and Content

Shapes



Recall: It's All About Shape and Content

Shapes



Shapes filled with content from a set $A = \{a_1, a_2, \ldots\}$

Recall: Natural Functors on Set

Set = the class of all sets

Recall: Natural Functors on Set

$F : \mathsf{Set} \to \mathsf{Set}$ is a natural functor if:

Recall: Natural Functors on Set

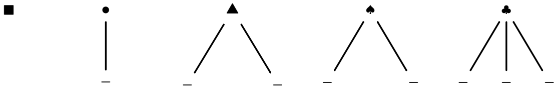
$F : \mathsf{Set} \to \mathsf{Set}$ is a natural functor if:

It comes with a set of shapes

Recall: Natural Functors on Set

$F : \text{Set} \to \text{Set}$ is a natural functor if:

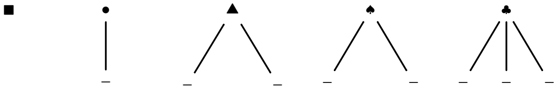
It comes with a set of shapes, say



Recall: Natural Functors on Set

F : Set → Set is a natural functor if:

It comes with a set of shapes, say



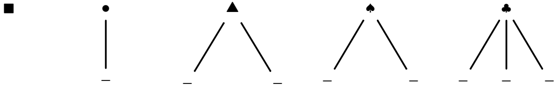
Each element $x \in \mathsf{F}\,A$ consists of:

a choice of a shape

Recall: Natural Functors on Set

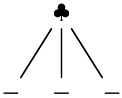
$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$ is a natural functor if:

It comes with a set of shapes, say



Each element $x \in \mathsf{F}\ A$ consists of:

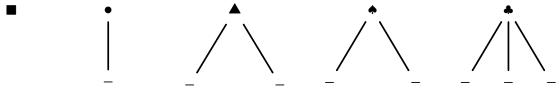
a choice of a shape, say



Recall: Natural Functors on Set

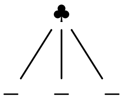
F : Set → Set is a natural functor if:

It comes with a set of shapes, say



Each element $x \in \mathsf{F}\, A$ consists of:

a choice of a shape, say

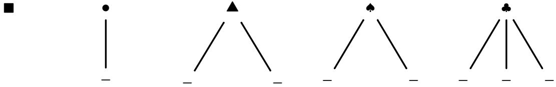


a filling with content from A

Recall: Natural Functors on Set

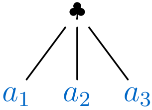
$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$ is a natural functor if:

It comes with a set of shapes, say



Each element $x \in \mathsf{F}\,A$ consists of:

a choice of a shape, say



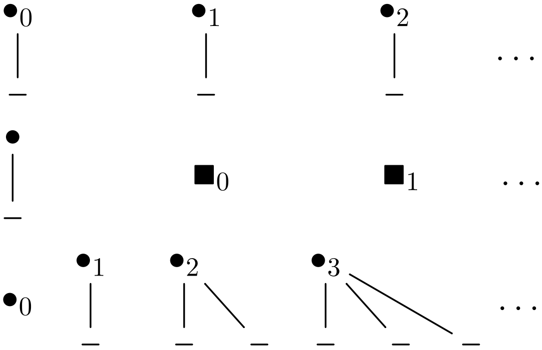
a filling with content from A, say

Recall: Examples of Natural Functors

$\mathsf{F}\ A = \mathbb{N} \times A$

$\mathsf{F}\ A = \mathbb{N} + A$

$\mathsf{F}\ A = \mathsf{List}\ A$

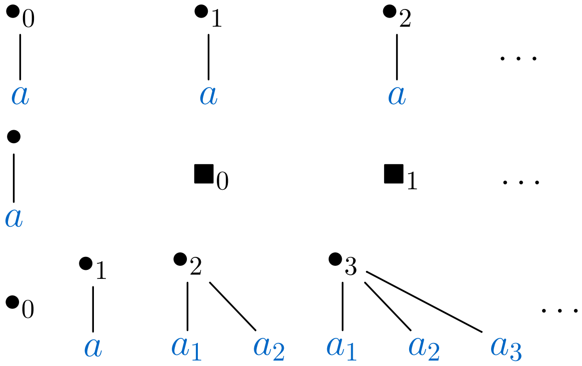


Recall: Examples of Natural Functors

$\mathsf{F}\ A = \mathbb{N} \times A$

$\mathsf{F}\ A = \mathbb{N} + A$

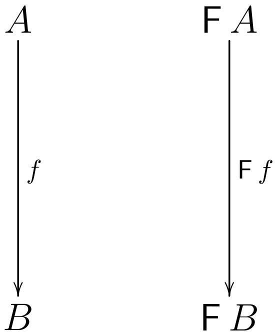
$\mathsf{F}\ A = \mathsf{List}\ A$



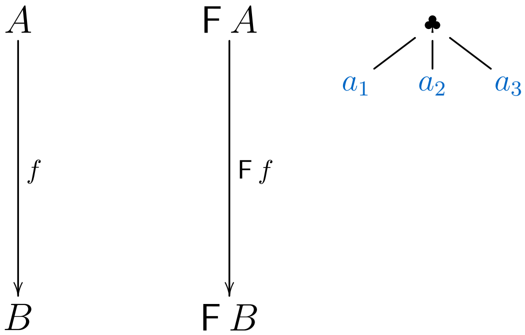
Functorial Action (Mapper)

$$\begin{array}{c} A \\ \Big\downarrow f \\ B \end{array}$$

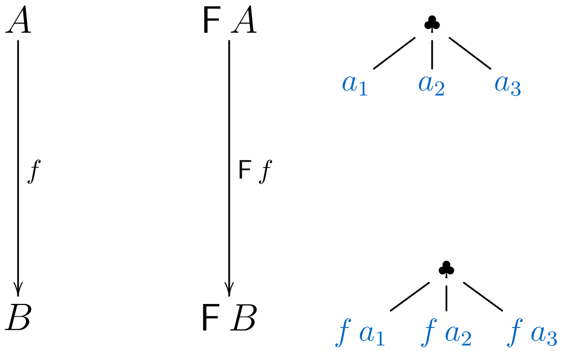
Functorial Action (Mapper)



Functorial Action (Mapper)



Functorial Action (Mapper)



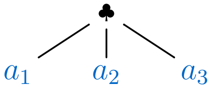
Keep the same shape

Apply f to the content

Atoms

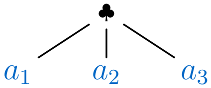
$$\mathsf{F}\,A \xrightarrow{\mathsf{Fatoms}_A} \mathcal{P}\,A$$

Atoms



$$\mathsf{F}\,A \xrightarrow{\mathsf{Fatoms}_A} \mathcal{P}\,A$$

Atoms



$$\{a_1, a_2, a_3\}$$

$$\mathsf{F}\,A \xrightarrow{\mathsf{Fatoms}_A} \mathcal{P}\,A$$

Natural Functors

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Functoriality: For all $A \xrightarrow{f} B$, we have $\mathsf{F}\,A \overset{\mathsf{F}\,f}{\Rightarrow} \mathsf{F}\,B$ such that:

$$\mathsf{F}\ \mathsf{id}_A = \mathsf{id}_{\mathsf{F}\,A}$$
$$\mathsf{F}\,(g \circ f) = \mathsf{F}\,g \circ \mathsf{F}\,f$$

Naturality: For all A, we have $\mathsf{F}\,A \overset{\mathsf{Fatoms}_A}{\Rightarrow} \mathcal{P}\,A$ such that, for all $A \xrightarrow{f} B$:

$$\mathsf{image}\ f \circ \mathsf{Fatoms}_A = \mathsf{Fatoms}_B \circ \mathsf{image}\ f$$

Examples

$$A \overset{f}{\Rightarrow} B \qquad \mathsf{F}\ A \overset{\mathsf{F}\,f}{\Rightarrow} \mathsf{F}\ B \qquad \mathsf{F}\ A \overset{\mathsf{Fatoms}}{\Rightarrow} \mathcal{P}\ A$$

$\mathsf{F}\ A = \mathbb{N} \times A$

$$\mathsf{F} f\ (n, a) = (n, f\ a)$$
$$\mathsf{Fatoms}\ (n, a) = \{a\}$$

$\mathsf{F}\ A = \mathbb{N} + A$

$$\mathsf{F} f\ (\mathsf{Left}\ n) = \mathsf{Left}\ n \qquad \mathsf{F} f\ (\mathsf{Right}\ a) = \mathsf{Right}\ (f\ a)$$
$$\mathsf{Fatoms}\ (\mathsf{Left}\ n) = \varnothing \qquad \mathsf{Fatoms}\ (\mathsf{Right}\ a) = \{a\}$$

$\mathsf{F}\ A = \mathsf{List}\ A$

$$\mathsf{F} f\ (a_1 \cdot a_2 \cdot \ldots \cdot a_n) = f\ a_1 \cdot f\ a_2 \ldots \cdot f\ a_n$$
$$\mathsf{Fatoms}\ (a_1 \cdot a_2 \cdot \ldots \cdot a_n) = \{a_1, a_2, \ldots, a_n\}$$

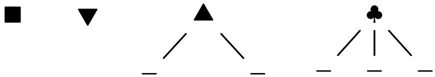
Recall: Iterating Shape Composition

Natural functor $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Recall: Iterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

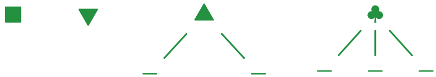
The shapes of F:



Recall: Iterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

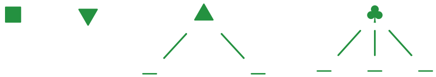
Copies of the shapes of F:



Recall: Iterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

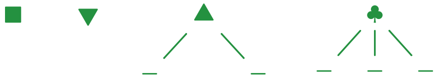
Copies of the shapes of F:



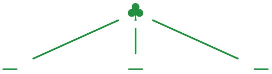
Put them together by plugging in shape for content slot

Recall: Iterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

Copies of the shapes of F: 

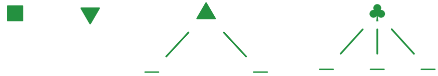
Put them together by plugging in shape for content slot



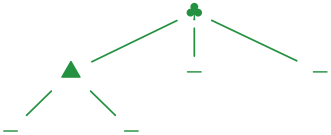
Recall: Iterating Shape Composition

Natural functor $F : Set \to Set$

Copies of the shapes of F:



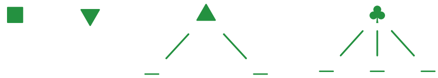
Put them together by plugging in shape for content slot



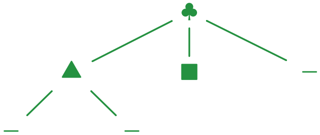
Recall: Iterating Shape Composition

Natural functor $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



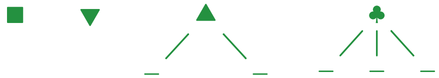
Put them together by plugging in shape for content slot



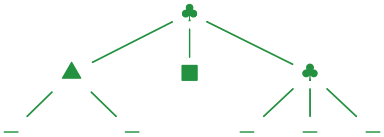
Recall: Iterating Shape Composition

Natural functor $F : Set \to Set$

Copies of the shapes of F:



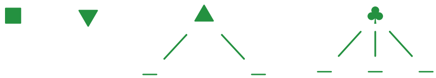
Put them together by plugging in shape for content slot



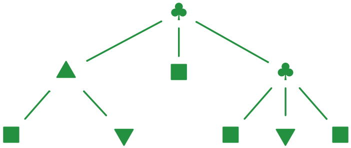
Recall: Iterating Shape Composition

Natural functor $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



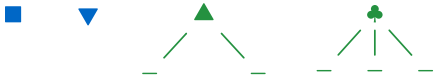
Put them together by plugging in shape for content slot until there are no lingering slots left!



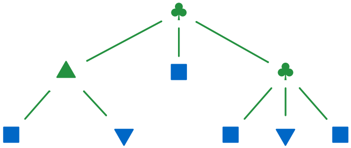
Recall: Iterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

Copies of the shapes of F:



Put them together by plugging in shape for content slot until there are no lingering slots left!

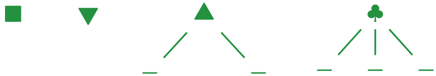


The leaves are always empty-content shapes

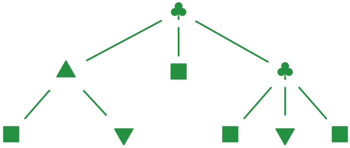
Recall: Iterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



Put them together by plugging in shape for content slot
until there are no lingering slots left!



Define I_F = the set of all such finitary couplings

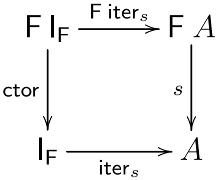
Recall: Properties of $\mathsf{I_F}$

Given a natural functor F, $(\mathsf{I_F},\ \mathsf{ctor} : \mathsf{F}\ \mathsf{I_F} \to \mathsf{I_F})$ satisfies:

ctor bijection

$\mathsf{I_F}$ = the datatype of F

Iteration (Initial Algebra Property): For all $(A, s : \mathsf{F}\ A \to A)$, there exists a unique function iter_s such that



Induction: Given any predicate φ on $\mathsf{I_F}$

$$\frac{\forall x \in \mathsf{F}\ \mathsf{I_F}.\ (\forall \mathsf{i} \in \mathsf{Fatoms}\ \mathsf{x}.\ \varphi\ \mathsf{i}) \Rightarrow \varphi\ (\mathsf{ctor}\ \mathsf{x})}{\forall i \in \mathsf{I_F}.\ \varphi\ \mathsf{i}}$$

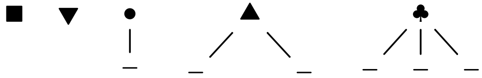
Coiterating Shape Composition

Natural functor $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

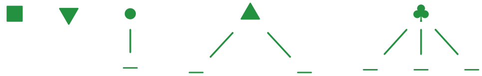
The shapes of F:



Coiterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

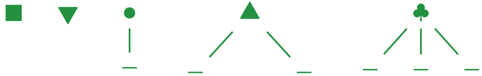
Copies of the shapes of F:



Coiterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

Copies of the shapes of F:

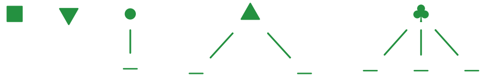


Put them together by plugging in shape for content slot

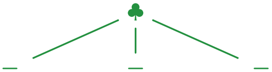
Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:

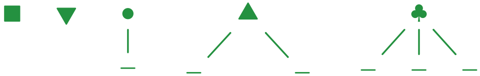


Put them together by plugging in shape for content slot

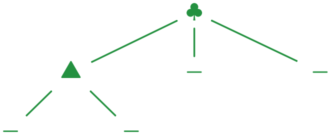


Coiterating Shape Composition

Natural functor F : Set → Set

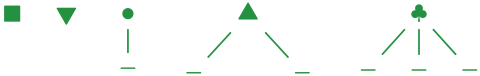
Copies of the shapes of F: 

Put them together by plugging in shape for content slot

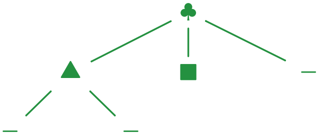


Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

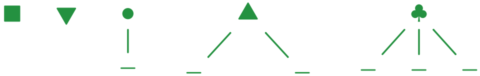
Copies of the shapes of F: 

Put them together by plugging in shape for content slot

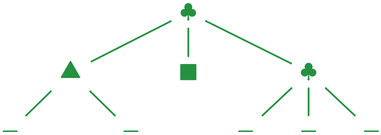


Coiterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

Copies of the shapes of F: 

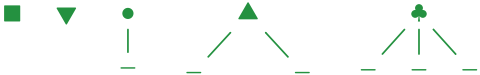
Put them together by plugging in shape for content slot



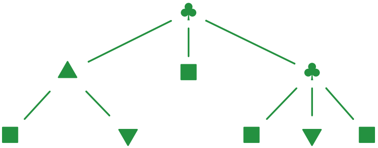
Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



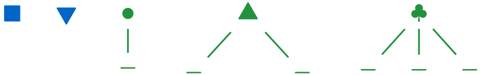
Put them together by plugging in shape for content slot until there are no lingering slots left!



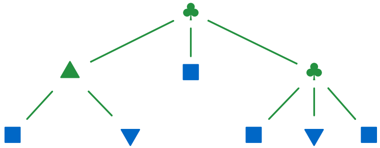
Coiterating Shape Composition

Natural functor $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



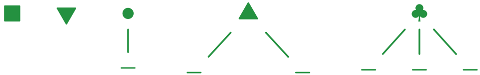
Put them together by plugging in shape for content slot until there are no lingering slots left!



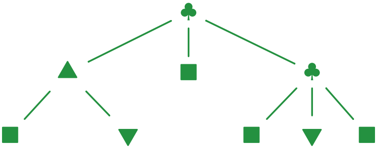
The leaves are always empty-content shapes

Coiterating Shape Composition

Natural functor $F : \mathrm{Set} \to \mathrm{Set}$

Copies of the shapes of F: 

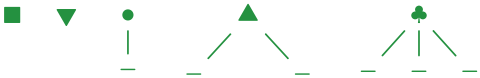
Put them together by plugging in shape for content slot until there are no lingering slots left!



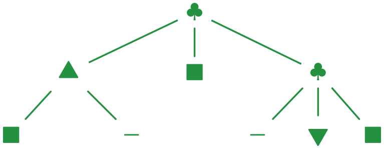
The leaves are always empty-content shapes

Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F: 

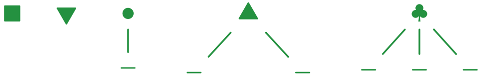
Put them together by plugging in shape for content slot
until there are no lingering slots left!



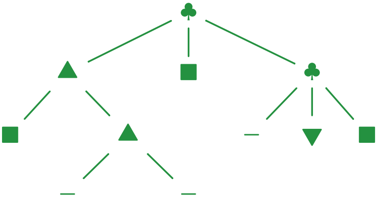
Allow infinite couplings

Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F: 

Put them together by plugging in shape for content slot
until there are no lingering slots left!

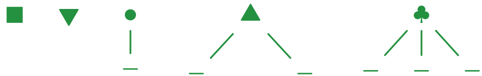


Allow infinite couplings

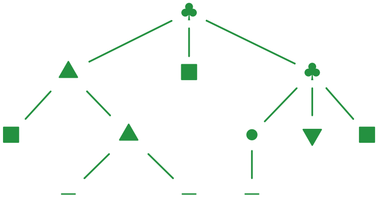
Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



Put them together by plugging in shape for content slot
until there are no lingering slots left!

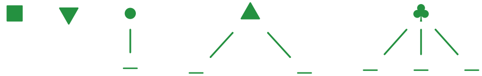


Allow infinite couplings

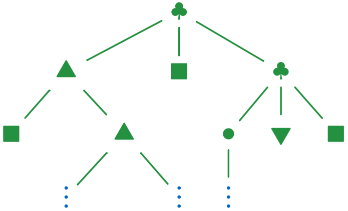
Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:



Put them together by plugging in shape for content slot until there are no lingering slots left!

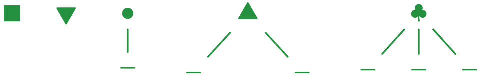


Allow infinite couplings

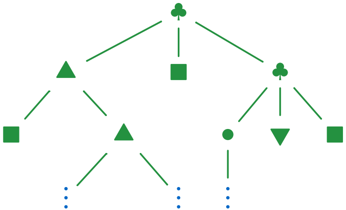
Coiterating Shape Composition

Natural functor $F : \mathsf{Set} \to \mathsf{Set}$

Copies of the shapes of F:

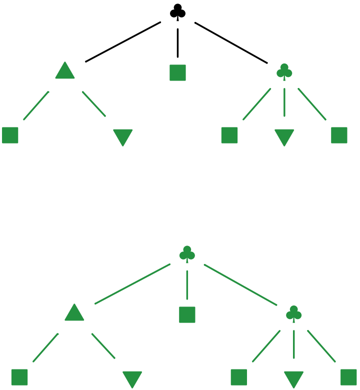


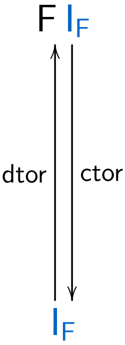
Put them together by plugging in shape for content slot until there are no lingering slots left!



Define J_F = the set of all such (possibly) infinitary couplings

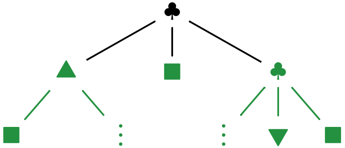
Recall: Properties of I_F: Bijectivity

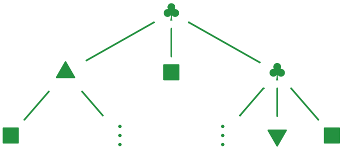


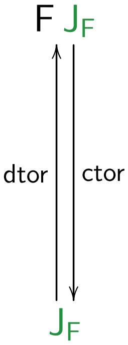


ctor and dtor are mutually inverse bijections

Properties of J_F: Bijectivity

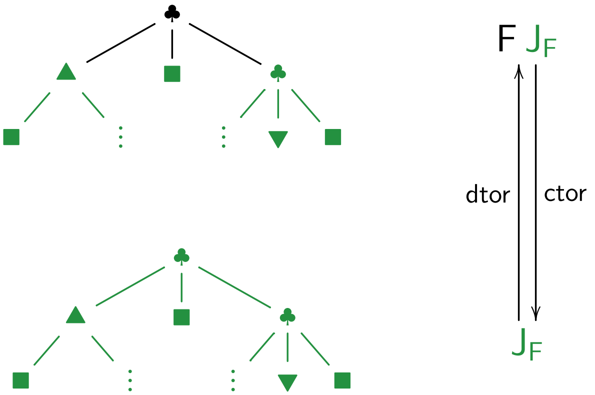






ctor and dtor are mutually inverse bijections

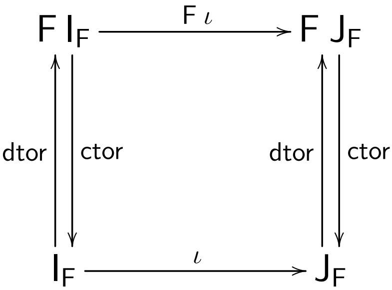
Properties of J_F: Bijectivity



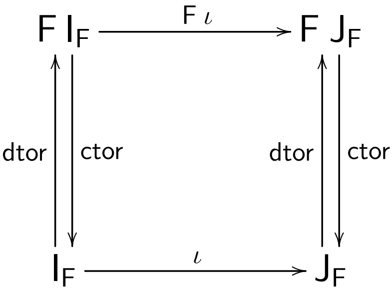
ctor and dtor are mutually inverse bijections

A similar property holds for J_F, where we use the same notations for constructor and destructor

I_F is embedded in J_F

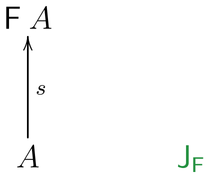


I_F is embedded in J_F

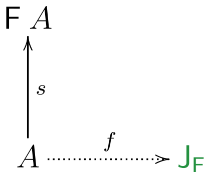


$\iota = \text{iter}_{\text{ctor}:F\ J_F \to F\ J_F}$

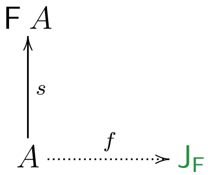
Properties of $\mathsf{J_F}$: Coiteration



Properties of $\mathsf{J_F}$: Coiteration

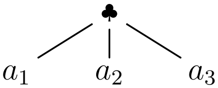


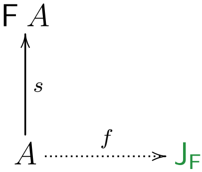
Properties of J_F: Coiteration



a

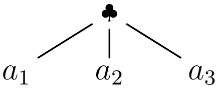
Properties of $\mathsf{J_F}$: Coiteration

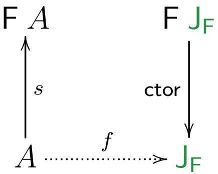




a

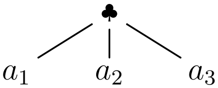
Properties of $\mathsf{J_F}$: Coiteration

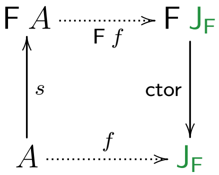




a

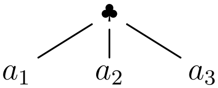
Properties of $\mathsf{J_F}$: Coiteration

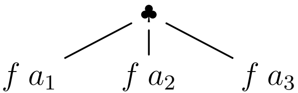


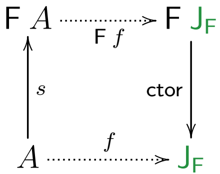


a

Properties of $\mathsf{J_F}$: Coiteration

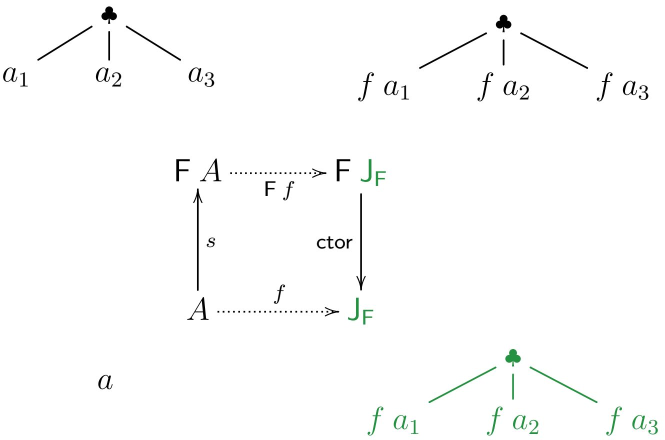




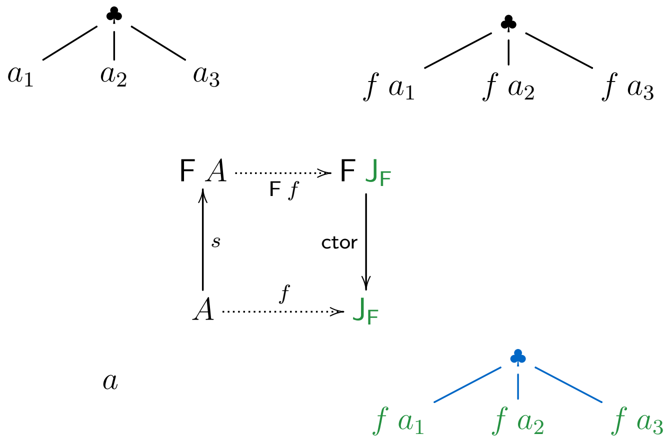


a

Properties of J_F: Coiteration

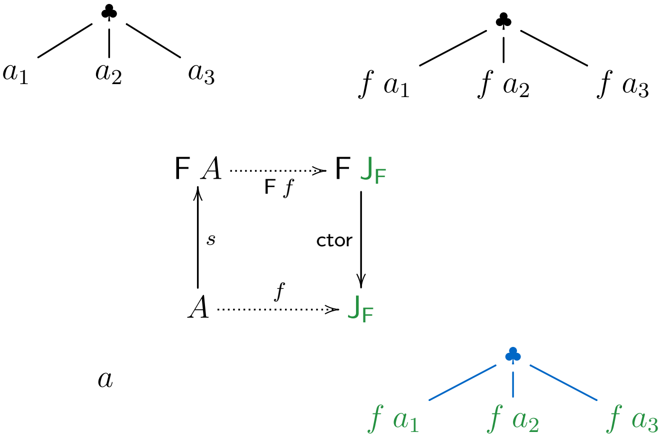


Properties of $\mathsf{J_F}$: Coiteration



a_1, a_2, a_3 are not “smaller” than a in any sense

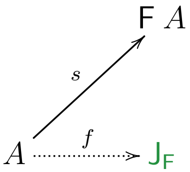
Properties of $\mathsf{J_F}$: Coiteration



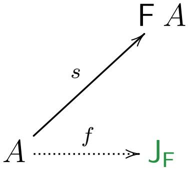
a_1, a_2, a_3 are not “smaller” than a in any sense

But computation has made progress

Properties of $\mathsf{J_F}$: Coiteration



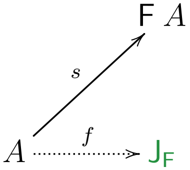
Properties of $\mathsf{J_F}$: Coiteration



a

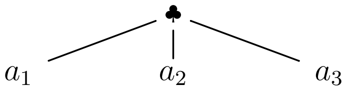
Properties of $\mathsf{J_F}$: Coiteration

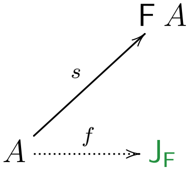
$s\ a$



a

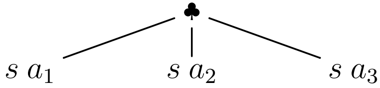
Properties of J_F: Coiteration

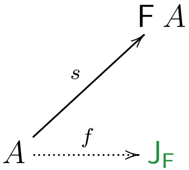




a

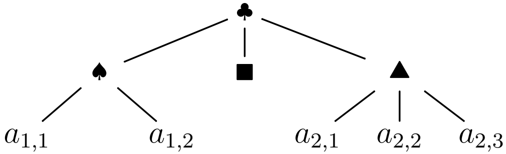
Properties of $\mathsf{J_F}$: Coiteration

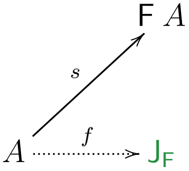




a

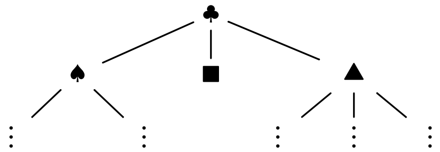
Properties of $\mathsf{J_F}$: Coiteration

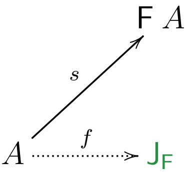




a

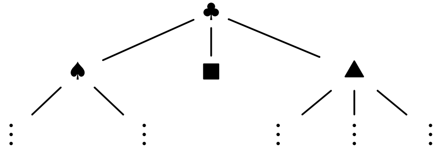
Properties of $\mathsf{J_F}$: Coiteration

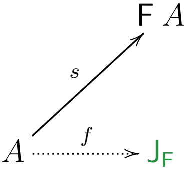




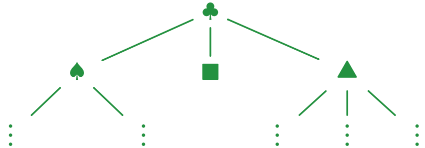
a

Properties of $\mathsf{J_F}$: Coiteration

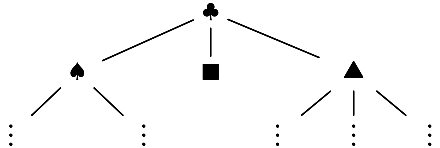


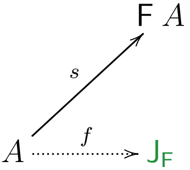


a



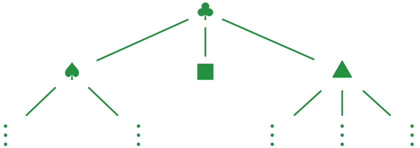
Properties of $\mathsf{J_F}$: Coiteration





$s\ a$ = the seed encoding the growth of the tree $f\ a$

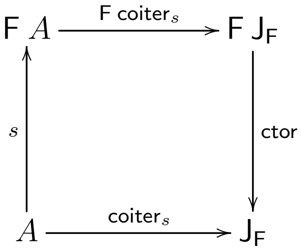
a



Properties of $\mathsf{J_F}$: Coiteration

Given a natural functor F, $(\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$

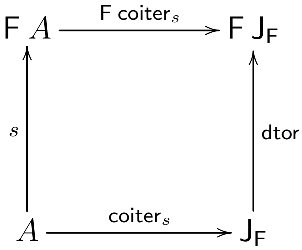
Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



Properties of $\mathsf{J_F}$: Coiteration

Given a natural functor F, $(\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$

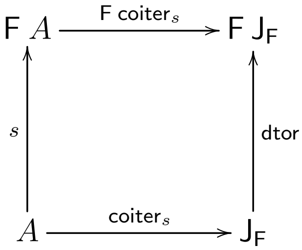
Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



Properties of $\mathsf{J_F}$: Coiteration

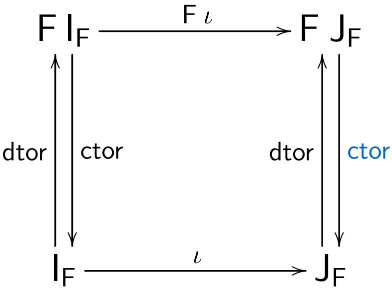
Given a natural functor F, $\quad (\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$

Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



$\mathsf{J_F}$ = the codatatype of F

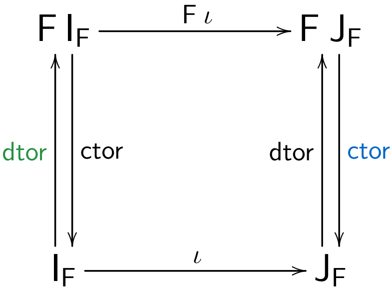
The I_F to J_F embedding revisited



ι can be regarded as defined by iteration on I_F

$\iota = \mathrm{iter}_{\mathrm{ctor}}$

The I_F to J_F embedding revisited



ι can be regarded as defined by
iteration on I_F but also by coiteration on J_F!

$\iota = \text{iter}_{\text{ctor}} = \text{coiter}_{\text{dtor}}$

Properties of J_F: Coinduction

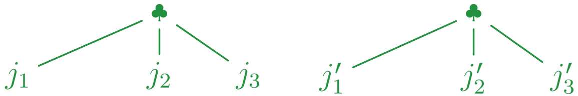
j j'

Properties of $\mathsf{J_F}$: Coinduction

j $\qquad$ j'

Want: $\quad j = j'$

Properties of J_F: Coinduction



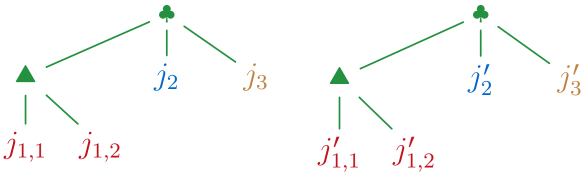
Want: $j = j'$

Properties of J_F: Coinduction

Suffices:

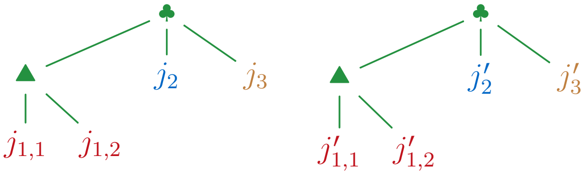
$$j_1 = j_1'$$
$$j_2 = j_2'$$
$$j_3 = j_3'$$

Properties of J_F: Coinduction



Suffices: $j_1 = j'_1$
$j_2 = j'_2$
$j_3 = j'_3$

Properties of J_F: Coinduction

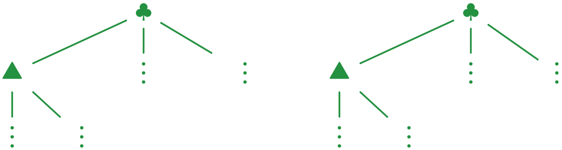


Suffices: $j_{1,1} = j'_{1,1}, \; j_{1,2} = j'_{1,2}$

$$j_2 = j'_2$$

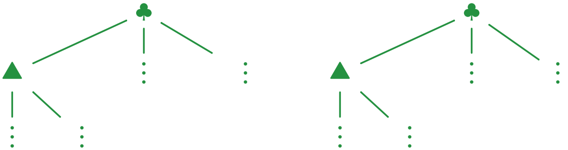
$$j_3 = j'_3$$

Properties of $\mathsf{J_F}$: Coinduction



Suffices: $j_{1,1} = j'_{1,1},\ j_{1,2} = j'_{1,2}$
$j_2 = j'_2$
$j_3 = j'_3$

Properties of J_F: Coinduction



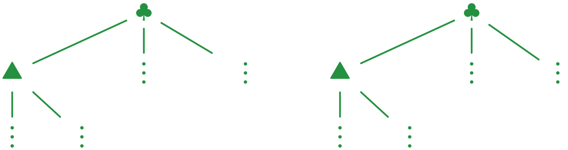
If we can stay in the game indefinitely, then equality holds!

Suffices: $j_{1,1} = j'_{1,1},\ j_{1,2} = j'_{1,2}$

$$j_2 = j'_2$$

$$j_3 = j'_3$$

Properties of J_F: Coinduction



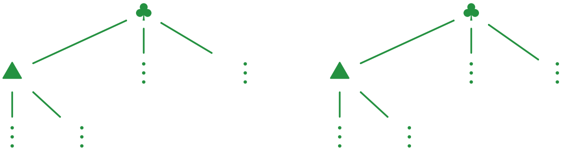
If we can stay in the game indefinitely, then equality holds!
But how to show we can "stay in the game"?

Suffices: $j_{1,1} = j'_{1,1},\ j_{1,2} = j'_{1,2}$

$$j_2 = j'_2$$

$$j_3 = j'_3$$

Properties of J_F: Coinduction



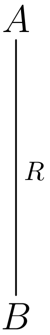
If we can stay in the game indefinitely, then equality holds!
But how to show we can "stay in the game"?
By exhibiting a "strategy"

Suffices: $j_{1,1} = j'_{1,1},\ j_{1,2} = j'_{1,2}$

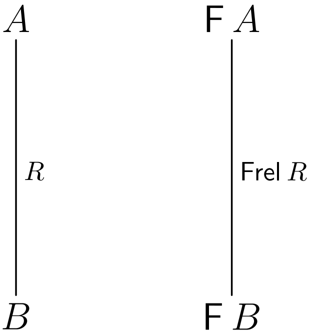
$$j_2 = j'_2$$

$$j_3 = j'_3$$

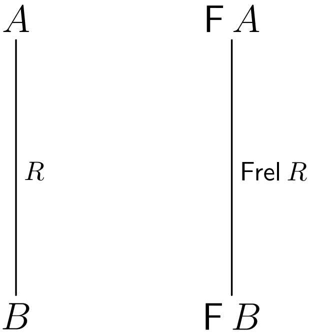
But First: Relators



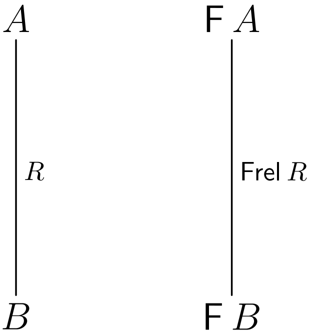
But First: Relators



But First: Relators

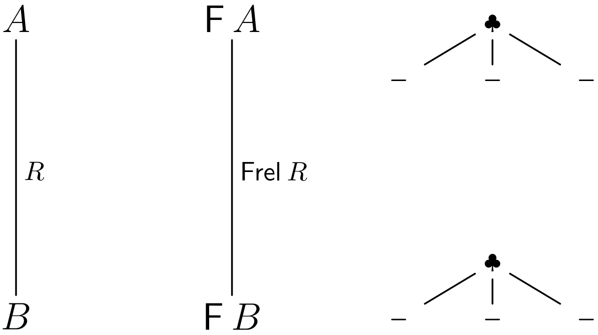


But First: Relators



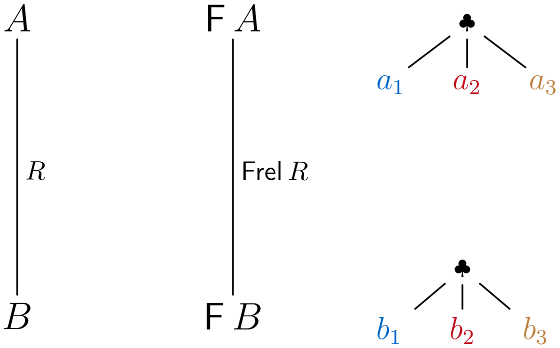
Two elements of F A and F B are related by Frel R iff

But First: Relators



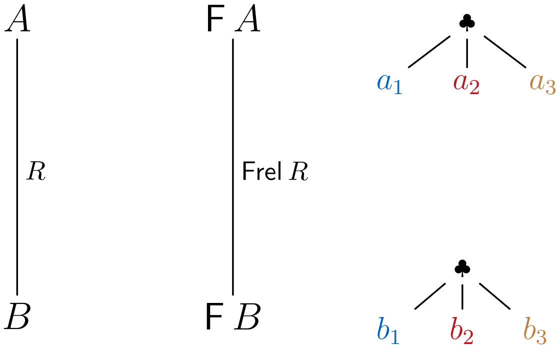
Two elements of F A and F B are related by Frel R iff they have the same shape

But First: Relators



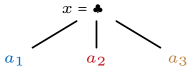
Two elements of F A and F B are related by Frel R iff
they have the same shape
and the contents from corresponding slots are related by R

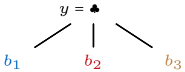
But First: Relators



Two elements of F A and F B are related by Frel R iff
they have the same shape
and the contents from corresponding slots are related by R
$R\ a_1\ b_1$, $R\ a_2\ b_2$, $R\ a_3\ b_3$

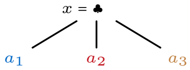
Relator Defined from Mapper

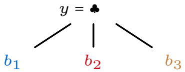




R relation between A and B, $x \in \mathsf{F}\ A$, $y \in \mathsf{F}\ B$

Relator Defined from Mapper

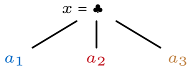


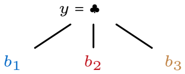


R relation between A and B, $x \in \mathsf{F}\ A$, $y \in \mathsf{F}\ B$

Frel $R\ x\ y$ defined as

Relator Defined from Mapper

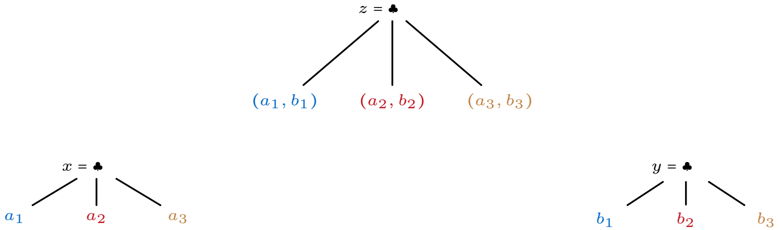




R relation between A and B, $x \in \mathsf{F}\ A$, $y \in \mathsf{F}\ B$

Frel $R\ x\ y$ defined as
$\exists z \in \mathsf{F}\ \{(a, b) \mid R\ a\ b\}.\ \mathsf{F}\ \pi_1\ z = x \wedge \mathsf{F}\ \pi_2\ z = y$

Relator Defined from Mapper



R relation between A and B, $x \in \mathsf{F}\ A$, $y \in \mathsf{F}\ B$

Frel $R\ x\ y$ defined as
$\exists z \in \mathsf{F}\ \{(a,b) \mid R\ a\ b\}.\ \mathsf{F}\ \pi_1\ z = x \wedge \mathsf{F}\ \pi_2\ z = y$

Relators for the Running Examples

R relation between A and B

Relators for the Running Examples

R relation between A and B

Frel R relation between F A and F B

Relators for the Running Examples

R relation between A and B

Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow$

Relators for the Running Examples

R relation between A and B

$\mathsf{Frel}\ R$ relation between $\mathsf{F}\ A$ and $\mathsf{F}\ B$

$$\mathsf{F}\ A = \mathbb{N} \times A \qquad \mathsf{Frel}\ R\ (m, a)\ (n, b) \Leftrightarrow (m = n\ \wedge\ R\ a\ b)$$

Relators for the Running Examples

R relation between A and B
Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow (m = n \land R\ a\ b)$

F $A = \mathbb{N} + A$

Relators for the Running Examples

R relation between A and B
Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow (m = n \wedge R\ a\ b)$

Frel $R\ u\ v \Leftrightarrow$

F $A = \mathbb{N} + A$

Relators for the Running Examples

R relation between A and B

$\mathsf{Frel}\ R$ relation between $\mathsf{F}\ A$ and $\mathsf{F}\ B$

$\mathsf{F}\ A = \mathbb{N} \times A \qquad \mathsf{Frel}\ R\ (m, a)\ (n, b) \Leftrightarrow (m = n \ \wedge\ R\ a\ b)$

$\mathsf{F}\ A = \mathbb{N} + A$

$$\begin{aligned}&\mathsf{Frel}\ R\ u\ v \Leftrightarrow \\ &\quad (\exists n.\ u = v = \mathsf{Left}\ n) \vee \\ &\quad (\exists a, b.\ u = \mathsf{Right}\ a \ \wedge\ v = \mathsf{Right}\ b \ \wedge\ R\ a\ b)\end{aligned}$$

Relators for the Running Examples

R relation between A and B

Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow (m = n\ \wedge\ R\ a\ b)$

F $A = \mathbb{N} + A$

Frel $R\ u\ v \Leftrightarrow$

$(\exists n.\ u = v = \text{Left}\ n) \vee$

$(\exists a, b.\ u = \text{Right}\ a\ \wedge\ v = \text{Right}\ b\ \wedge\ R\ a\ b)$

F A = List A

Relators for the Running Examples

R relation between A and B
Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow (m = n\ \wedge\ R\ a\ b)$

F $A = \mathbb{N} + A$ Frel $R\ u\ v \Leftrightarrow$
$(\exists n.\ u = v = \text{Left}\ n) \vee$
$(\exists a, b.\ u = \text{Right}\ a\ \wedge\ v = \text{Right}\ b\ \wedge\ R\ a\ b)$

F $A = \text{List}\ A$ Frel $R\ (a_1 \cdot a_2 \cdot \ldots \cdot a_m)\ \ (b_1 \cdot b_2 \cdot \ldots \cdot b_n) \Leftrightarrow$

Relators for the Running Examples

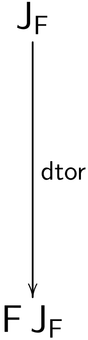
R relation between A and B
Frel R relation between F A and F B

F $A = \mathbb{N} \times A$ Frel $R\ (m, a)\ (n, b) \Leftrightarrow (m = n\ \wedge\ R\ a\ b)$

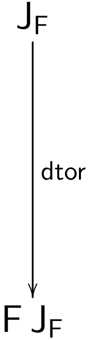
F $A = \mathbb{N} + A$ Frel $R\ u\ v \Leftrightarrow$
$(\exists n.\ u = v = \text{Left}\ n) \vee$
$(\exists a, b.\ u = \text{Right}\ a\ \wedge\ v = \text{Right}\ b\ \wedge\ R\ a\ b)$

F $A = \text{List}\ A$ Frel $R\ (a_1 \cdot a_2 \cdot \ldots \cdot a_m)\ (b_1 \cdot b_2 \cdot \ldots \cdot b_n) \Leftrightarrow$
$m = n\ \wedge\ (\forall i.\ R\ a_i\ b_i)$

Back to the “Strategy” for Proving Equality

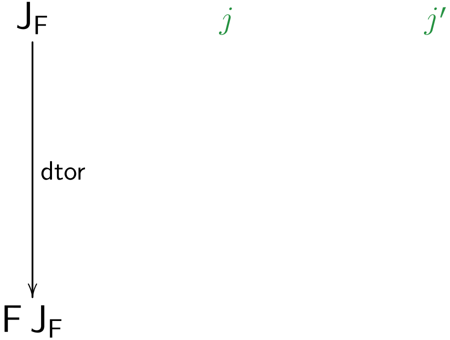


Back to the “Strategy” for Proving Equality



Given binary relation R on J_F

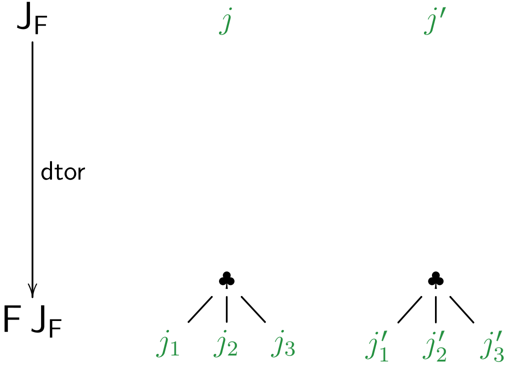
Back to the “Strategy” for Proving Equality



Given binary relation R on J_F

If $\forall j, j'.\ R\ j\ j'$

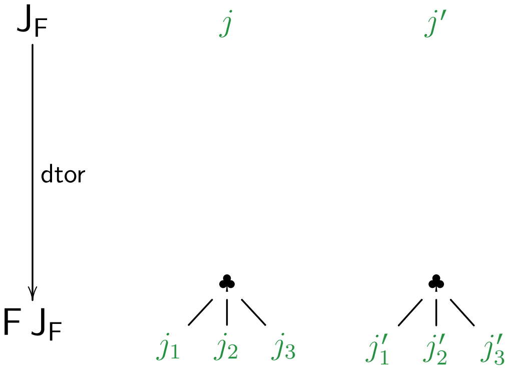
Back to the "Strategy" for Proving Equality



Given binary relation R on $\mathsf{J_F}$

If $\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')$

Back to the “Strategy” for Proving Equality

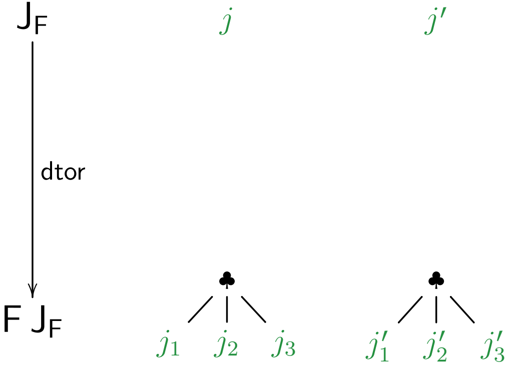


Given binary relation R on $\mathsf{J_F}$

If $\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')$

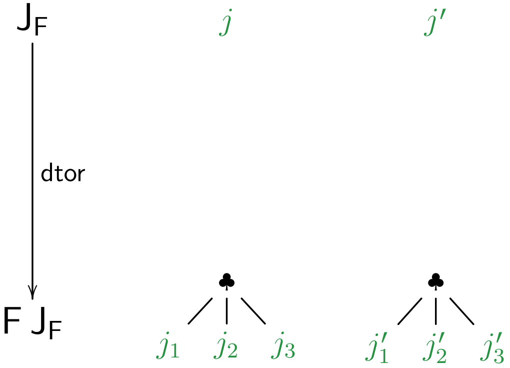
Then R is included in equality

Back to the “Strategy” for Proving Equality



Given binary relation R on $\mathsf{J_F}$
If $\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')$
Then R is included in equality $\quad \forall j, j'.\ R\ j\ j' \Rightarrow j = j'$

Back to the "Strategy" for Proving Equality

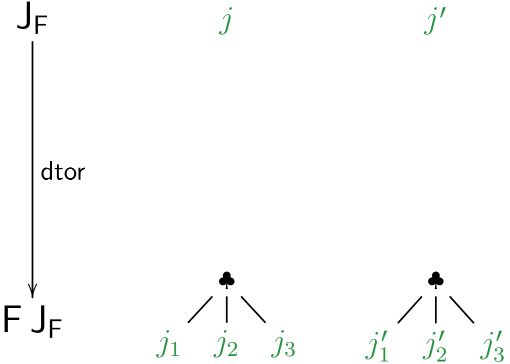


Given binary relation R on J_F

If $\forall j, j'.\ R\ j\ j' \Rightarrow \text{Frel}\ R\ (\text{dtor}\ j)\ (\text{dtor}\ j')$ | R F-bisimulation |

Then R is included in equality $\quad \forall j, j'.\ R\ j\ j' \Rightarrow j = j'$

Back to the "Strategy" for Proving Equality



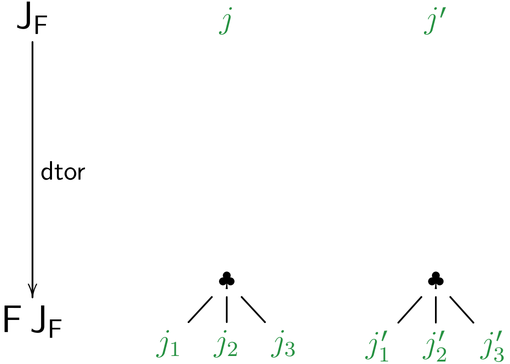
Summary: to prove $j = j'$,

Given binary relation R on J_F

If $\forall j, j'.\ R\ j\ j' \Rightarrow \text{Frel}\ R\ (\text{dtor}\ j)\ (\text{dtor}\ j')$ $\boxed{R \text{ F-bisimulation}}$

Then R is included in equality $\quad \forall j, j'.\ R\ j\ j' \Rightarrow j = j'$

Back to the “Strategy” for Proving Equality



Summary: to prove $j = j'$, find F-bisimulation R with $R\ j\ j'$

Given binary relation R on $\mathsf{J_F}$

If $\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')$ R F-bisimulation

Then R is included in equality $\forall j, j'.\ R\ j\ j' \Rightarrow j = j'$

Summary for J_F

Given a natural functor F, $(J_F,\ \text{dtor} : J_F \to F\ J_F)$ satisfies:

Summary for J_F

Given a natural functor F, $(J_F,\ \mathrm{dtor} : J_F \to F\ J_F)$ satisfies:

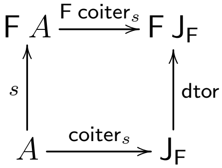
dtor bijection

Summary for $\mathsf{J_F}$

Given a natural functor F, $(\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$ satisfies:

dtor bijection

Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with

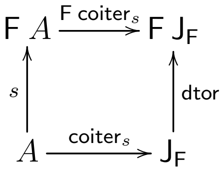


Summary for J_F

Given a natural functor F, $(\mathsf{J}_\mathsf{F},\ \mathsf{dtor} : \mathsf{J}_\mathsf{F} \to \mathsf{F}\ \mathsf{J}_\mathsf{F})$ satisfies:

dtor bijection

Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



Coinduction: Given any binary relation R on J_F

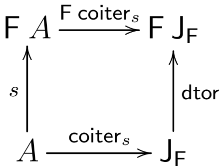
$$\frac{R \text{ is an F-bisimulation}}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Summary for $\mathsf{J_F}$

Given a natural functor F, $(\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$ satisfies:

dtor bijection

Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



Coinduction: Given any binary relation R on $\mathsf{J_F}$

$$\frac{\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

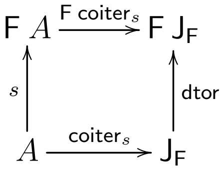
Summary for $\mathsf{J_F}$

Given a natural functor F, $\quad (\mathsf{J_F},\ \mathsf{dtor} : \mathsf{J_F} \to \mathsf{F}\ \mathsf{J_F})$ satisfies:

dtor bijection

$\mathsf{J_F}$ = the codatatype of F

Coiteration (Final Coalgebra Property): For all $(A, s : A \to \mathsf{F}\ A)$, there exists a unique function coiter_s with



Coinduction: Given any binary relation R on $\mathsf{J_F}$

$$\frac{\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

The shapes of F:

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

Or, graphically:

for each $b \in B$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

Or, graphically:  for each $b \in B$

Who is J_{F}?

Example of Codatatype

Let B be a fixed set. $\quad \mathsf{F}\, A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

Or, graphically:  for each $b \in B$

Who is $\mathsf{J_F}$?

Its elements have the form $(b_1, (b_2, \ldots, (b_n, \ldots$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\ A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

Or, graphically:  for each $b \in B$

Who is J_{F}?
Its elements have the form $(b_1, (b_2, \ldots, (b_n, \ldots$
I.e., essentially streams $b_1 \cdot b_2 \cdot \ldots \cdot b_n \cdot \ldots$

Example of Codatatype

Let B be a fixed set. $\mathsf{F}\,A = B \times A$

The shapes of F: $(b, \_)$ for each $b \in B$

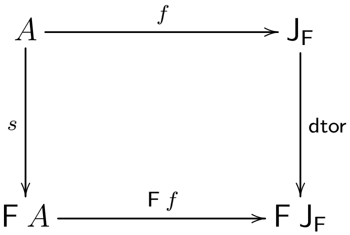
Or, graphically:

for each $b \in B$

Who is $\mathsf{J_F}$?
Its elements have the form $(b_1, (b_2, \ldots, (b_n, \ldots$
I.e., essentially streams $b_1 \cdot b_2 \cdot \ldots \cdot b_n \cdot \ldots$
So $\mathsf{J_F} = \mathsf{Stream_B}$

Example of Codatatype: Stream

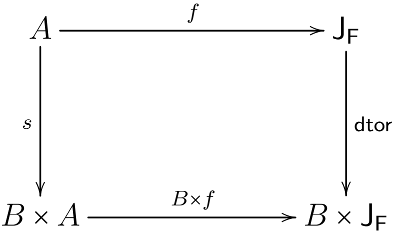
B fixed $\mathsf{F}\ A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$



$\mathsf{dtor}\ (f\ a) = (\mathsf{F}\ f)\ (s\ a)$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$

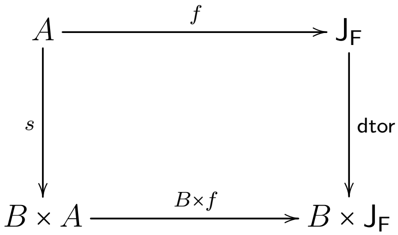


$\mathsf{dtor}\ (f\ a) = (\mathsf{F}\ f)\ (s\ a)$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$

Define: $\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$
$\mathsf{hd}^A = \pi_1 \circ s$ $\mathsf{tl}^A = \pi_2 \circ s$



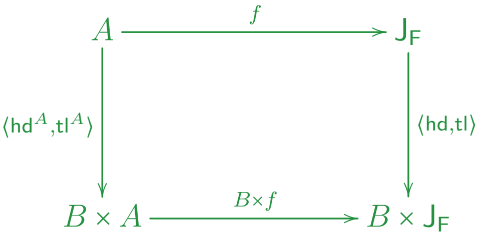
$\mathsf{dtor}\ (f\ a) = (\mathsf{F}\ f)\ (s\ a)$

Example of Codatatype: Stream

B fixed $\quad$ $\mathsf{F}\ A = B \times A$ $\quad$ $f = \mathsf{coiter}_s$ $\quad$ $\mathsf{J_F} = \mathsf{Stream_B}$

Define: $\mathsf{hd} = \pi_1 \circ \mathsf{dtor} \quad \mathsf{tl} = \pi_2 \circ \mathsf{dtor}$

$\mathsf{hd}^A = \pi_1 \circ s \quad \mathsf{tl}^A = \pi_2 \circ s$

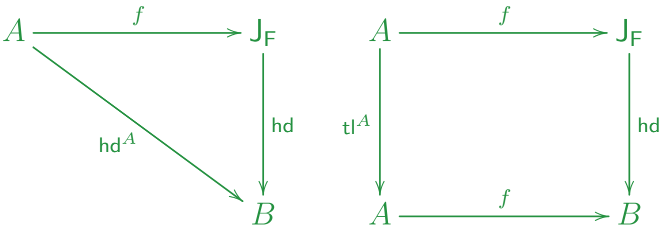


$\mathsf{dtor}\ (f\ a) = (\mathsf{F}\ f)\ (s\ a)$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$

Define: $\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$
$\mathsf{hd}^A = \pi_1 \circ s$ $\mathsf{tl}^A = \pi_2 \circ s$



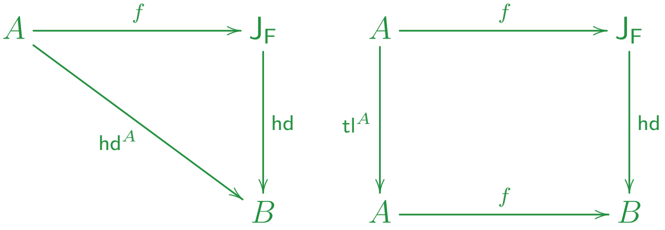
$\mathsf{dtor}\ (f\ a) = (\mathsf{F}\ f)\ (s\ a)$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$

Define: $\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$

$\mathsf{hd}^A = \pi_1 \circ s$ $\mathsf{tl}^A = \pi_2 \circ s$



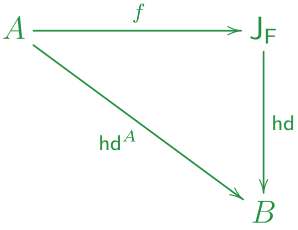
$\mathsf{hd}\ (f\ a) = \mathsf{hd}^A\ a$

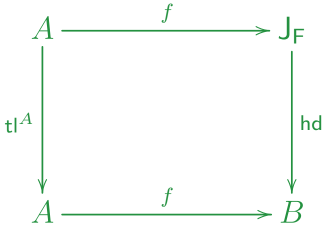
$\mathsf{tl}\ (f\ a) = f\ (\mathsf{tl}^A\ a)$

Example of Codatatype: Stream

B fixed $\mathsf{F}\, A = B \times A$ $f = \mathsf{coiter}_s$ $\mathsf{J_F} = \mathsf{Stream_B}$

Define: $\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$
$\mathsf{hd}^A = \pi_1 \circ s$ $\mathsf{tl}^A = \pi_2 \circ s$



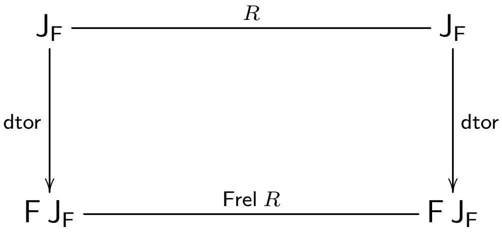


$\mathsf{hd}\,(f\ a) = \mathsf{hd}^A\ a$
$\mathsf{tl}\,(f\ a) = f\ (\mathsf{tl}^A\ a)$

Standard stream coiteration

Example of Codatatype: Stream

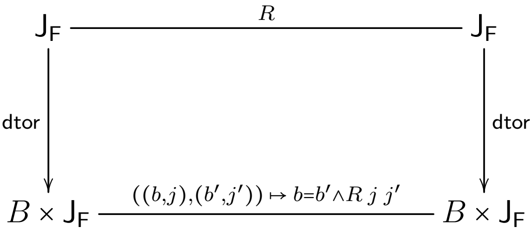
B fixed $\mathsf{F}\ A = B \times A$ $\mathsf{J_F} = \mathsf{Stream_B}$



$$\frac{R \text{ is an F-bisimulation}}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $\mathsf{J_F} = \mathsf{Stream_B}$

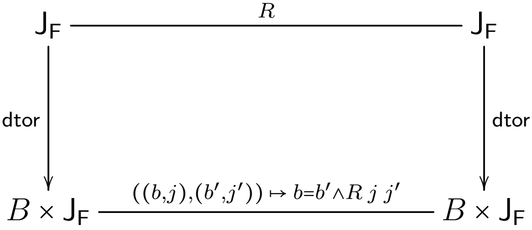


$$\frac{R \text{ is an F-bisimulation}}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $\mathsf{J_F} = \mathsf{Stream_B}$

$\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$

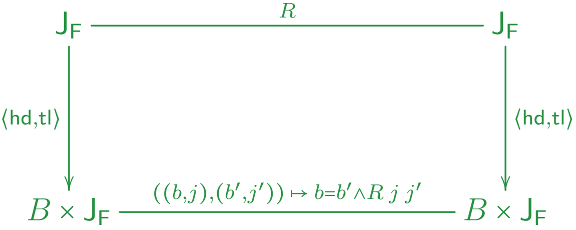


$$\frac{R \text{ is an F-bisimulation}}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed $\mathsf{F}\ A = B \times A$ $\mathsf{J_F} = \mathsf{Stream_B}$

$\mathsf{hd} = \pi_1 \circ \mathsf{dtor}$ $\mathsf{tl} = \pi_2 \circ \mathsf{dtor}$

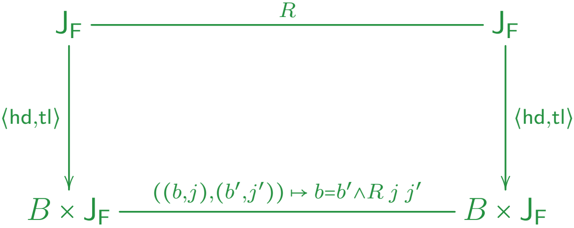


$$\frac{R \text{ is an F-bisimulation}}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed F A = $B \times A$ $\mathsf{J_F} = \mathsf{Stream_B}$

hd = $\pi_1 \circ$ dtor tl = $\pi_2 \circ$ dtor

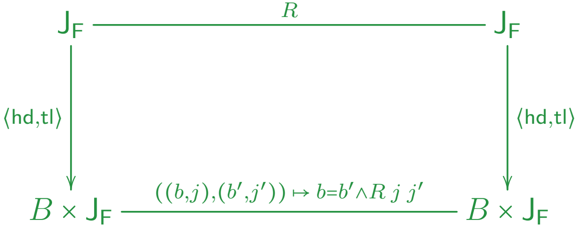


$$\frac{\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{dtor}\ j)\ (\mathsf{dtor}\ j')}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed $\quad$ $\mathsf{F}\ A = B \times A$ $\quad$ $\mathsf{J_F} = \mathsf{Stream_B}$

$\mathsf{hd} = \pi_1 \circ \mathsf{dtor} \quad \mathsf{tl} = \pi_2 \circ \mathsf{dtor}$

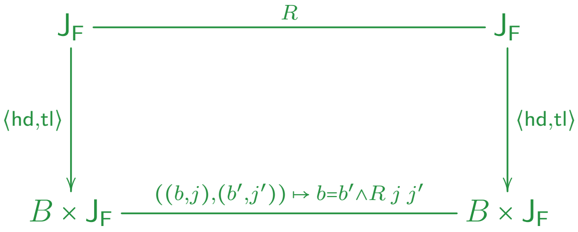


$$\frac{\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{Frel}\ R\ (\mathsf{hd}\ j, \mathsf{tl}\ j)\ (\mathsf{hd}\ j', \mathsf{tl}\ j')}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Example of Codatatype: Stream

B fixed $\quad$ $\mathsf{F}\ A = B \times A$ $\qquad$ $\mathsf{J_F} = \mathsf{Stream_B}$

$\mathsf{hd} = \pi_1 \circ \mathsf{dtor} \quad \mathsf{tl} = \pi_2 \circ \mathsf{dtor}$



$$\frac{\forall j, j'.\ R\ j\ j' \Rightarrow \mathsf{hd}\ j = \mathsf{hd}\ j' \ \wedge\ R\ (\mathsf{tl}\ j)\ (\mathsf{tl}\ j')}{\forall j, j'.\ R\ j\ j' \Rightarrow j = j'}$$

Concrete Example of Coiteration

$$\begin{aligned} &\mathsf{ev} : \mathsf{Stream}_B \to \mathsf{Stream}_B \\ &\quad \mathsf{hd}\ (\mathsf{ev}\ j) = \mathsf{hd}\ j \\ &\quad \mathsf{tl}\ (\mathsf{ev}\ j) = \mathsf{ev}\ (\mathsf{tl}\ (\mathsf{tl}\ j)) \end{aligned}$$

Concrete Example of Coiteration

$\mathsf{ev} : \mathsf{Stream}_B \to \mathsf{Stream}_B$

$\quad \mathsf{hd}\,(\mathsf{ev}\ j) = \mathsf{hd}\ j$

$\quad \mathsf{tl}\,(\mathsf{ev}\ j) = \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))$

$\mathsf{odd} : \mathsf{Stream}_B \to \mathsf{Stream}_B$

$\quad \mathsf{hd}\,(\mathsf{odd}\ j) = \mathsf{hd}\,(\mathsf{tl}\ j)$

$\quad \mathsf{tl}\,(\mathsf{odd}\ j) = \mathsf{odd}\,(\mathsf{tl}\,(\mathsf{tl}\ j))$

Concrete Example of Coiteration

$\mathsf{ev} : \mathsf{Stream}_B \to \mathsf{Stream}_B$

$\quad \mathsf{hd}\,(\mathsf{ev}\ j) = \mathsf{hd}\ j$

$\quad \mathsf{tl}\,(\mathsf{ev}\ j) = \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))$

$\mathsf{odd} : \mathsf{Stream}_B \to \mathsf{Stream}_B$

$\quad \mathsf{hd}\,(\mathsf{odd}\ j) = \mathsf{hd}\,(\mathsf{tl}\ j)$

$\quad \mathsf{tl}\,(\mathsf{odd}\ j) = \mathsf{odd}\,(\mathsf{tl}\,(\mathsf{tl}\ j))$

$\mathsf{zip} : \mathsf{Stream}_B \times \mathsf{Stream}_B \to \mathsf{Stream}_B$

$\quad \mathsf{hd}\,(\mathsf{zip}\,(j_1, j_2)) = \mathsf{hd}\ j_1$

$\quad \mathsf{tl}\,(\mathsf{zip}\,(j_1, j_2)) = \mathsf{zip}\,(j_2,\ \mathsf{tl}\ j_1)$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

$\mathsf{tl}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{tl}\ j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{hd}\ j$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

$\mathsf{tl}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{tl}\ j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{hd}\ j$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\;j, \mathsf{odd}\;j) = j$

$\mathsf{zip}\,(\mathsf{odd}\;j,\; \mathsf{tl}\,(\mathsf{ev}\;j)) = \mathsf{tl}\;j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\;j, \mathsf{odd}\;j)) = \mathsf{hd}\;j$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\;j, \mathsf{odd}\;j) = j$

$\mathsf{zip}\,(\mathsf{odd}\;j,\;\mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\;j))) = \mathsf{tl}\;j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\;j, \mathsf{odd}\;j)) = \mathsf{hd}\;j$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

$\mathsf{zip}\,(\mathsf{odd}\ j,\ \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) = \mathsf{tl}\ j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{hd}\ j$

$\mathsf{tl}\,(\mathsf{zip}\,(\mathsf{odd}\ j,\ \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j)))) = \mathsf{tl}\,(\mathsf{tl}\ j)$

$\mathsf{hd}\ \ldots = \mathsf{hd}\,(\mathsf{tl}\ j)$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

$\mathsf{zip}\,(\mathsf{odd}\ j,\ \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) = \mathsf{tl}\ j$ $\quad$ $\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{hd}\ j$

$\mathsf{zip}\,(\mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j)), \mathsf{odd}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) = \mathsf{tl}\,(\mathsf{tl}\ j)$ $\quad$ $\mathsf{hd}\ \ldots = \mathsf{hd}\,(\mathsf{tl}\ j)$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\; j, \mathsf{odd}\; j) = j$

$\mathsf{zip}\,(\mathsf{odd}\; j,\; \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\; j))) = \mathsf{tl}\; j$ $\quad$ $\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\; j, \mathsf{odd}\; j)) = \mathsf{hd}\; j$

$\mathsf{zip}\,(\mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\; j)), \mathsf{odd}\,(\mathsf{tl}\,(\mathsf{tl}\; j))) = \mathsf{tl}\,(\mathsf{tl}\; j)$ $\quad$ $\mathsf{hd}\; \ldots = \mathsf{hd}\,(\mathsf{tl}\; j)$

Pattern-Based Incremental Coinduction

$\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j) = j$

$\mathsf{zip}\,(\mathsf{odd}\ j,\ \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) = \mathsf{tl}\ j$

$\mathsf{hd}\,(\mathsf{zip}\,(\mathsf{ev}\ j, \mathsf{odd}\ j)) = \mathsf{hd}\ j$

$\mathsf{zip}\,(\mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j)), \mathsf{odd}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) = \mathsf{tl}\,(\mathsf{tl}\ j)$

$\mathsf{hd}\ \ldots = \mathsf{hd}\,(\mathsf{tl}\ j)$

Bisimulation: $R\ j_1\ j_2 \equiv$
$j_1 = \mathsf{zip}\,(\mathsf{ev}\ j_2, \mathsf{odd}\ j_2) \lor$
$\exists j.\ j_1 = \mathsf{zip}\,(\mathsf{odd}\ j,\ \mathsf{ev}\,(\mathsf{tl}\,(\mathsf{tl}\ j))) \land j_2 = \mathsf{tl}\ j$

Universe of (Co)Datatypes

Natural functors are a class of functors

Universe of (Co)Datatypes

Natural functors are a class of functors
containing the standard basic functors: sum, product, etc.

Universe of (Co)Datatypes

Natural functors are a class of functors

- containing the standard basic functors: sum, product, etc.
- closed under the datatype and codatatype constructor

Universe of (Co)Datatypes

Natural functors are a class of functors
- containing the standard basic functors: sum, product, etc.
- closed under the datatype and codatatype constructor

E.g.: fixing B, List_B is the datatype of $A \mapsto \{*\} + B \times A$

Universe of (Co)Datatypes

Natural functors are a class of functors
- containing the standard basic functors: sum, product, etc.
- closed under the datatype and codatatype constructor

E.g.: fixing B, List_B is the datatype of $A \mapsto \{*\} + B \times A$
but $B \mapsto \mathsf{List}_B$ is also a natural functor

Universe of (Co)Datatypes

Natural functors are a class of functors
- containing the standard basic functors: sum, product, etc.
- closed under the datatype and codatatype constructor

E.g.: fixing B, List_B is the datatype of $A \mapsto \{*\} + B \times A$
but $B \mapsto \mathsf{List}_B$ is also a natural functor
and similarly for $B \mapsto \mathsf{Stream}_B$

Universe of (Co)Datatypes

Natural functors are a class of functors
  containing the standard basic functors: sum, product, etc.
  closed under the datatype and codatatype constructor

E.g.: fixing B, List_B is the datatype of $A \mapsto \{*\} + B \times A$
but $B \mapsto \mathsf{List}_B$ is also a natural functor
and similarly for $B \mapsto \mathsf{Stream}_B$

Nesting datatypes in codatatypes or vice versa
allows for modular specs of fancy data structures

Universe of (Co)Datatypes in Isabelle/HOL

The Isabelle system maintains a database of natural functors

Universe of (Co)Datatypes in Isabelle/HOL

The Isabelle system maintains a database of natural functors

User can write high-level specifications:

codatatype Stream A = Cons (hd : A) (tl : List A)

Universe of (Co)Datatypes in Isabelle/HOL

The Isabelle system maintains a database of natural functors

User can write high-level specifications:

codatatype Stream A = Cons (hd : A) (tl : List A)

In the background:

- Isabelle parses this into a natural functor: $B \mapsto B \times A$
- Then infers high-level principles for (co)recursion and (co)induction for Stream
- Finally, Stream is itself registered as a natural functor

Examples

datatype List A = Nil $|$ Cons A (List A)

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

Examples

datatype List A = Nil $|$ Cons A (List A)

codatatype Lazy_List A = Nil $|$ Cons A (List A)

datatype X A = Leaf A $|$ Node (X A) (X A)

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node $(X\ A)\ (X\ A)$

Examples

datatype List A = Nil $|$ Cons A (List A)

codatatype Lazy_List A = Nil $|$ Cons A (List A)

datatype BTree A = Leaf A $|$ Node $(X\ A)$ $(X\ A)$

datatype $X\ A$ = Node A (List $(X\ A)$)

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

datatype Tree A = Node A (List (Tree A))

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

datatype Tree A = Node A (List (Tree A))

finite-depths, finitely branching
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node $(X\ A)$ $(X\ A)$

datatype Tree A = Node A (List (Tree A))
finite-depths, infinitely branching
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

datatype Tree A = Node A (Lazy_List (Tree A))
finite-depths, infinitely branching
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

datatype Tree A = Node A (Lazy_List (Tree A))
infinite-depths, infinitely branching
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node $(X\ A)$ $(X\ A)$

codatatype Tree A = Node A (Lazy_List (Tree A))
infinite-depths, infinitely branching
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (Lazy_List (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (Countable_Set (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (Set$_k$ (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (Multi_Set (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (Fuzzy_Set (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (PLUG_YOUR_OWN (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

Examples

datatype List A = Nil | Cons A (List A)

codatatype Lazy_List A = Nil | Cons A (List A)

datatype BTree A = Leaf A | Node (X A) (X A)

codatatype Tree A = Node A (PLUG_YOUR_OWN (Tree A))
infinite-depths, infinitely branching unordered
A-labeled trees

- Show a set operator to be a (bounded) natural functor
- Register it
- Then Isabelle will allow nesting it in (co)datatype expressions

Examples

datatype X A =
    Elements (Finite_Set (X A))

Examples

datatype Hereditarily_Finite_Set A =
  Elements (Finite_Set (Hereditarily_Finite_Set A))

Examples

datatype Hereditarily_Finite_Set A =
  Elements (Finite_Set (Hereditarily_Finite_Set A))

... in the presence of the Foundation Axiom

Examples

codatatype Hereditarily_Finite_Set A =
Elements (Finite_Set (Hereditarily_Finite_Set A))

... in the presence of Aczel's Anti-Foundation Axiom

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

The proof assistant Isabelle/HOL represents this universe and makes it available to the users

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

The proof assistant Isabelle/HOL represents this universe and makes it available to the users with a lot of sugar to hide the category theory ☺

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

The proof assistant Isabelle/HOL represents this universe and makes it available to the users with a lot of sugar to hide the category theory ☺

But... the category theory in the background offers flexibility unprecedented in proof assistants or programming languages

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

The proof assistant Isabelle/HOL represents this universe and makes it available to the users with a lot of sugar to hide the category theory ☺

But... the category theory in the background offers flexibility unprecedented in proof assistants or programming languages

Moreover, the abstract constructions have very concrete intuitions

Summary

Datatypes and codatatypes have intuitive representations in terms of Shape and Content

They form a rich, extendable universe

The proof assistant Isabelle/HOL represents this universe and makes it available to the users with a lot of sugar to hide the category theory ☺

But... the category theory in the background offers flexibility unprecedented in proof assistants or programming languages

Moreover, the abstract constructions have very concrete intuitions

The abstract reality can be very concrete

Relevant Literature

Our work:

Natural functors:
LICS'12, ESOP'15
Flexible corecursion:
ICFP'15
Isabelle implementation:
ITP'14
Case study:
IJCAR'14,
Incremental coinduction:
FOSSACS'10
Non-free datatypes:
LICS'10, ICFP'11, CPP'13

Other people's work:

Isabelle/ZF codata (Paulson)
Containers
(Abbott, Altenkirch, Ghani)
Fibrations
(Hermida, Jacobs)
Flexible Corecursion
(Turi/Plotkin, Bartels, Jacobs,
Milius, Hinze, Atkey/McBride)
Flexible Coinduction
(Rot, Bonsangue, Rutten, Silva,
Roșu, Endrullis, Hendriks,
Hur/Dreyer/Vafeiadis)
Corecursion in MiniAgda, Agda
(Abel)