Leicestershire, 8 January 2014

A Coalgebraic Foundation for Coinductive Union Types

Marcello Bonsangue (LIACS)

D. Ancona (Genova Univ.), F. de Boer (CWI), J. Rot (LIACS), J. Rutten (CWI)

Leiden Institute of Advanced Computer Science
Research & Education

Algebraic data types

- A possibly recursive type involving sums and (labelled) products.

Finite list of integers	List = null + <elm: Int, nxt: List>
Finite trees	Tree = null + <elm: Nat, l: Tree, r: Tree>

- Inductively defined data structures
  - Finite but unbounded data
  - Data can be fully unfolded by a recursive program

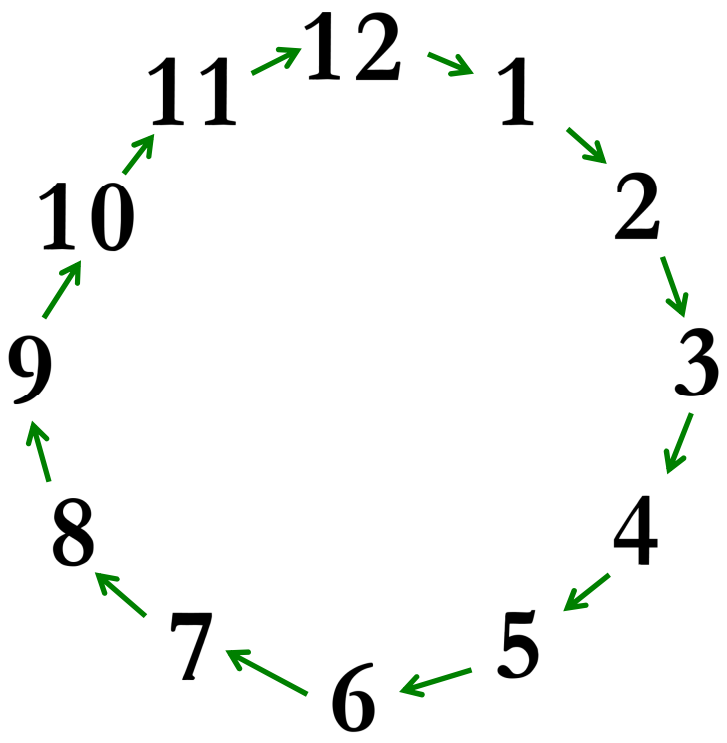
Coalgebraic data types

- A possibly recursive type involving sums and (labelled) products.

Infinite list of integers	List = <elm: Int, nxt: List>
Finite and infinite lists	List = null + <elm: Int, nxt: List>
Infinite trees	Tree = <elm: Nat, l: Tree, r: Tree>

- Coinductively defined data structures
  - Possibly infinite or circular data structures
  - Data can be manipulated while it is unfolded by a recursive program (possibly non-terminating)

Example: cyclic lists



- Inductive type **

  List = null + <elm: Int, nxt: List>

- Coinductive type

  List = <elm: Int, nxt: List>

** Accessing the *n*-th element of the list in a type safe way requires *n* useless non-null checks

Union types and subtyping

- Recent type systems rely on union and subtyping relations

Two approaches:

- Syntactic: define subtyping by axiomatizing it
  - Easy to forget a rule yielding sound but not complete systems
  - Not easily modifiable
- Semantics: types are sets of values, subtyping is set inclusion
  - Many properties are for free (e.g. transitivity)

In this talk:

semantics subtyping for coinductive data structures

Union types and subtyping

- Finite and infinite lists of Booleans and integers

  $X = \text{null} \vee (\text{Bool} \times X) \vee (\text{Int} \times X)$

- $Y = (\text{Bool} \times Z) \vee (\text{Bool} \times Y) \vee (\text{Int} \times Y)$ ends with a Boolean
- $V = (\text{Int} \times Z) \vee (\text{Bool} \times V) \vee (\text{Int} \times V)$ ends with an integer
- $Z = \text{null}$ just ends

- Intuitively $X <: Y \vee V \vee Z$ but how can we prove it?

Roadmap

- Plain coinductive types
- Coinductive union types for lists
- Coinductive union types for shapes

Type definitions

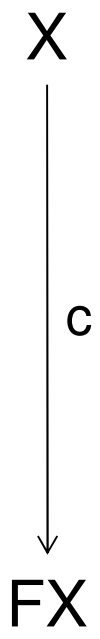
F-coalgebras are recursive type definitions

$FX = B + A \times X$

$x_1 = \langle a, x_2 \rangle$

$x_2 = b$

$x_3 = \langle a, x_3 \rangle$



Types

Elements of final F-coalgebra $\mathbb{T}$ are coinductive types

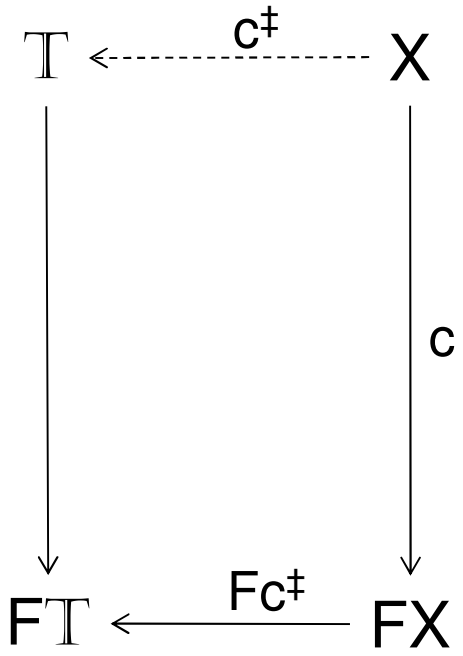
$FX = B + A \times X$

$\mathbb{T} = A^*B + A^\omega$

$c^\ddagger(x_1) = ab$

$c^\ddagger(x_2) = b$

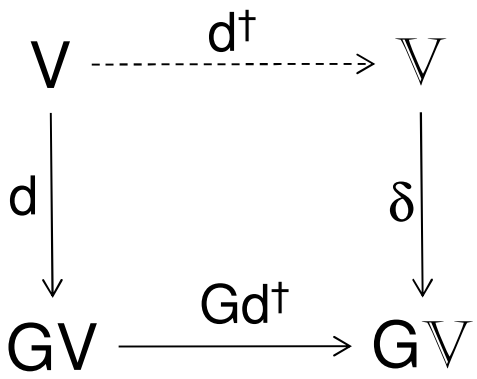
$c^\ddagger(x_3) = aaaaaaa\ldots$



Values and values definitions

G-coalgebras are recursive data definitions

Elements of final G-coalgebra $\mathbb{V}$ are coinductive data



Basic type assignment

α:G $\rightarrow$ F is a natural transformation mapping values to types

$GV = \mathbb{N} + \mathbb{N} \times V + \mathbb{B} \times V$

$FV = \{Nat\} + \{Nat,Bool\} \times V$

$\alpha(n) = Nat$ n:Nat

$\alpha(n,v) = \langle Nat,v \rangle$

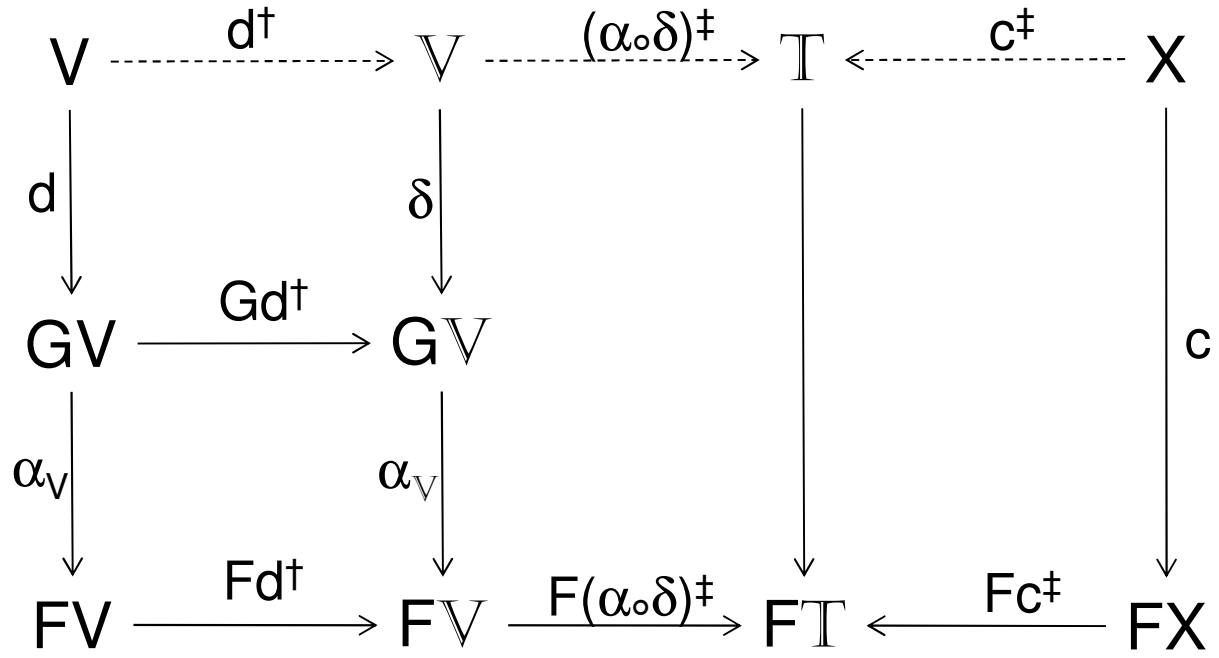
$\alpha(tt,v) = \langle Bool,v \rangle$

$\alpha(ff,v) = \langle Bool,v \rangle$

Type assignment

Type assignment $v{:}x$ iff $(\alpha \circ d)^\ddagger(v) = c^\ddagger(x)$

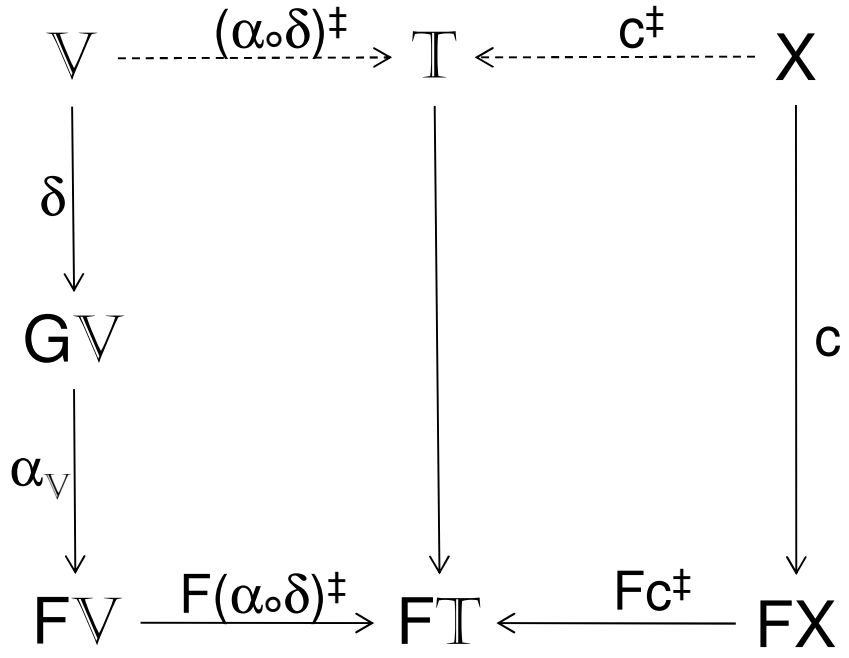
Theorem: If F preserves weak pullbacks then $v{:}x$ iff $v \sim_F x$



Types as set of values

$[\![-]\!] : \mathbb{T} \to \mathcal{P}(\mathbb{V})$

$[\![t]\!] = \{v \in \mathbb{V} \mid (\alpha \circ d)^\sharp(v) = t\}$



Soundness and completeness

If α is pointwise surjective and F preserves weak pullbacks then for all $x,y \in X$

$$[\![c^\sharp(x)]\!] = [\![c^\sharp(y)]\!]$$

iff

$$c^\sharp(x) = c^\sharp(y)$$

iff

$$x \sim_F y$$

Roadmap

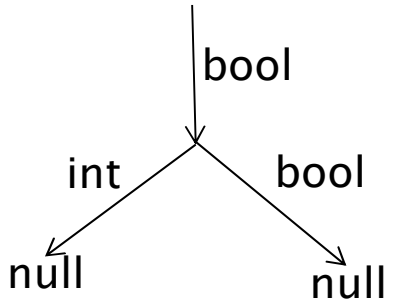
- Plain coinductive types
- Coinductive union types for lists
- Coinductive union types for shapes

First attempt

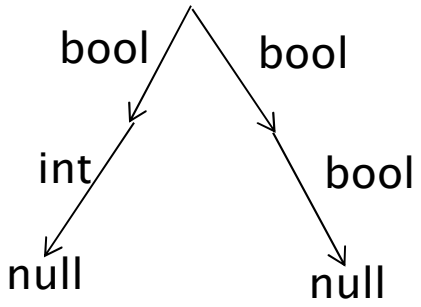
- Take a functor F for plain types
- Add unions by composing with the powerset functor $\mathcal{P}$
- Example:
  - finite and infinite lists — $FX = B+A\times X$
  - Final coalgebra — $\mathbb{T} = A^*B + A^\omega$
  - Union types — $\mathcal{P}FX = \mathcal{P}(B+A\times X)$

Bisimulation vs. traces

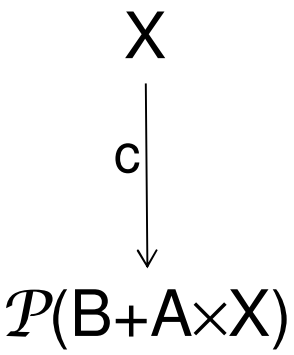
- $y_1 = \langle bool, y_2 \rangle$
- $y_2 = \langle int, y_3 \rangle \vee \langle bool, y_3 \rangle$
- $y_3 = null$

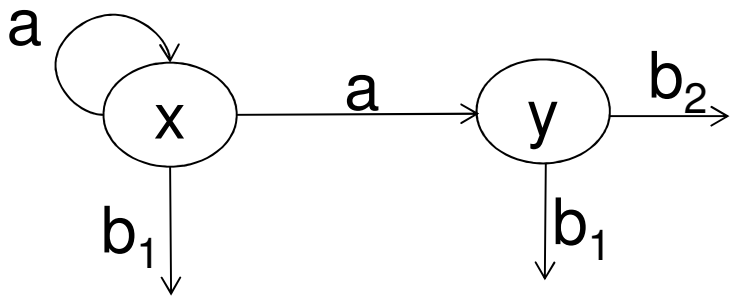


- $z_1 = \langle bool, z_2 \rangle \vee \langle bool, z_3 \rangle$
- $z_2 = \langle bool, y_3 \rangle$
- $z_3 = \langle int, y_3 \rangle$



Non-deterministic Moore Automata

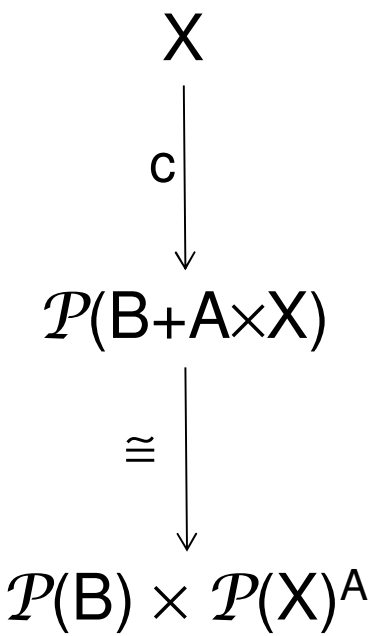




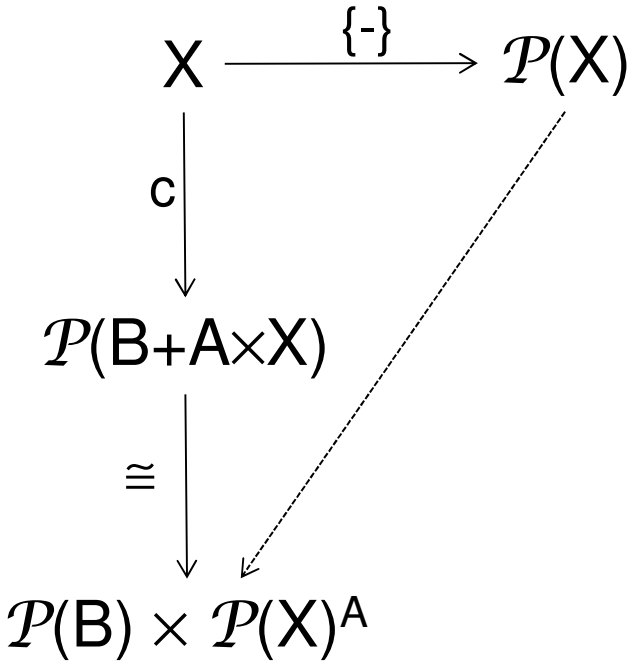
$x = \langle a, x \rangle \vee \langle a, y \rangle \vee b_1$

$y = b_1 \vee b_2$

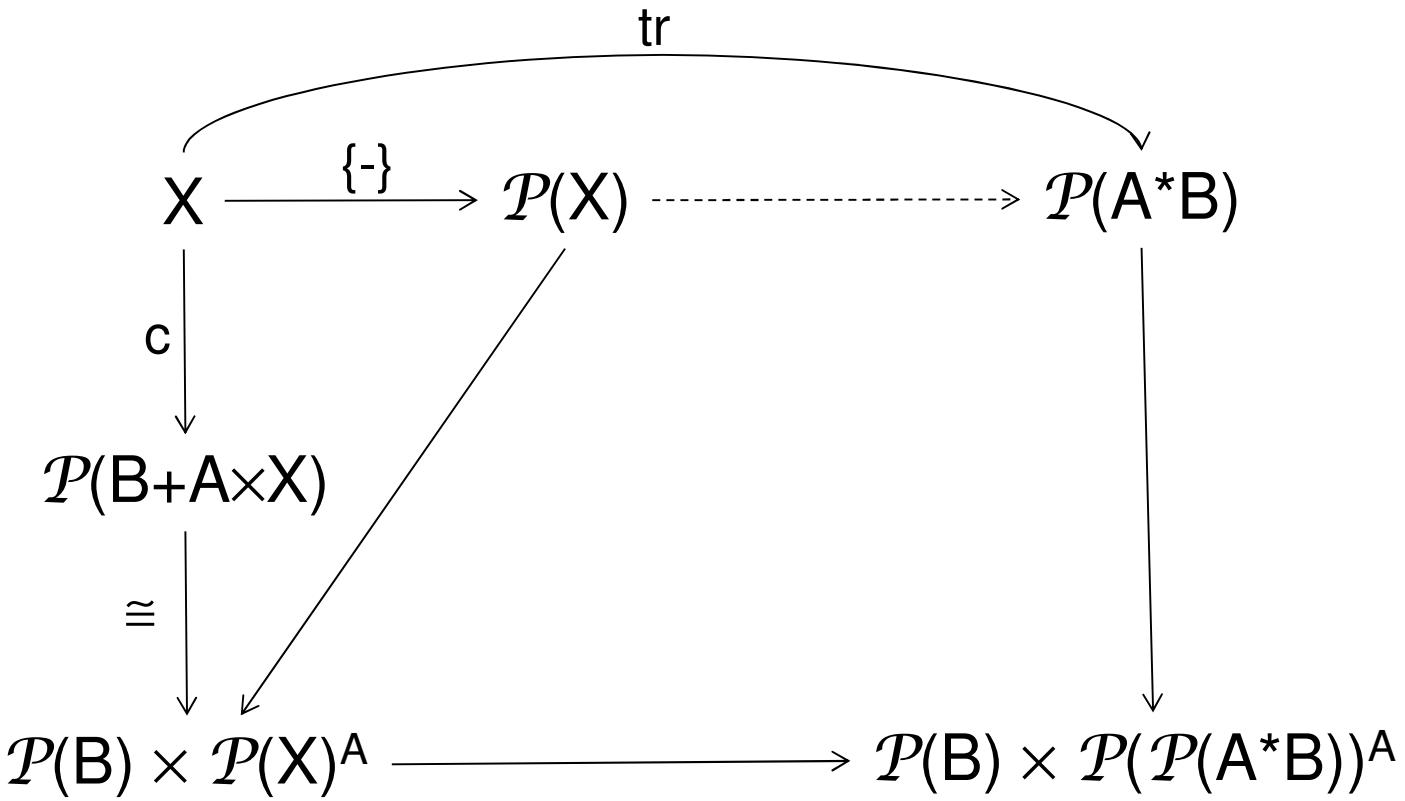
Trace semantics via determinization



Trace semantics via determinization

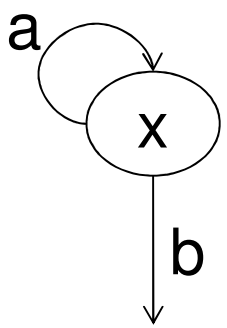


Trace semantics via determinization



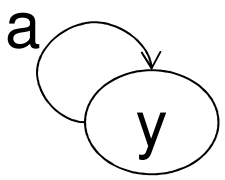
Trace semantics is not the intended one

$x = \langle a, x \rangle \vee b$



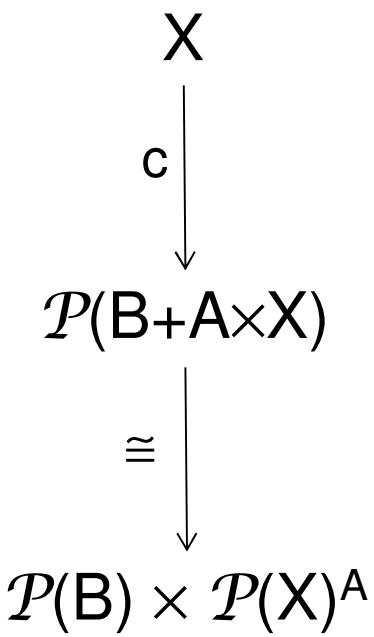
tr(x) = {b,ab,aab,aaab,…}

$y = \langle a, y \rangle$



tr(y) = ∅

Maximal traces

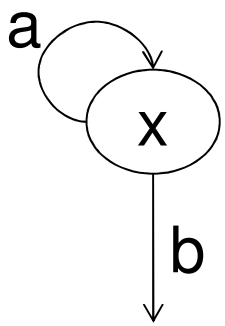


T(x) is the greatest subset of $\mathcal{P}(A^*B \cup A^\omega)$ such that

1. $b \in T(x)$ iff $b \in c(x)$
2. $aw \in T(x)$ iff $\exists y.\ \langle a,y\rangle \in c(x)$ & $w \in T(y)$

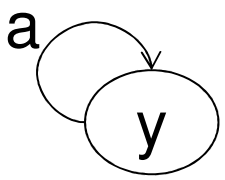
Maximal traces

$x = \langle a, x \rangle \vee b$



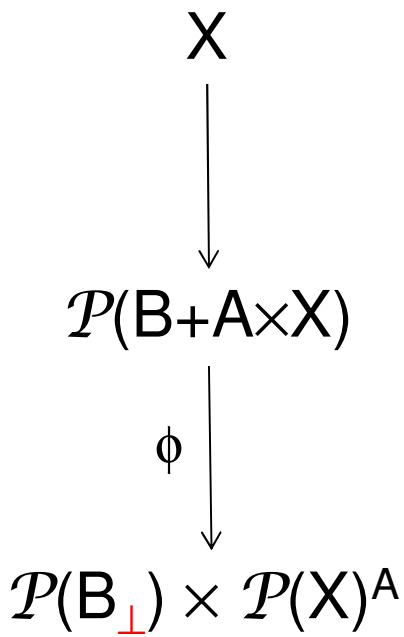
$T(x) = \{b, ab, aab, aaab, \ldots, a^{\omega}\}$

$y = \langle a, y \rangle$

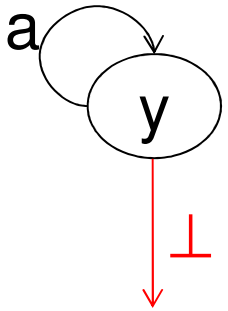


$T(y) = \{a^{\omega}\}$

Maximal traces via approximation

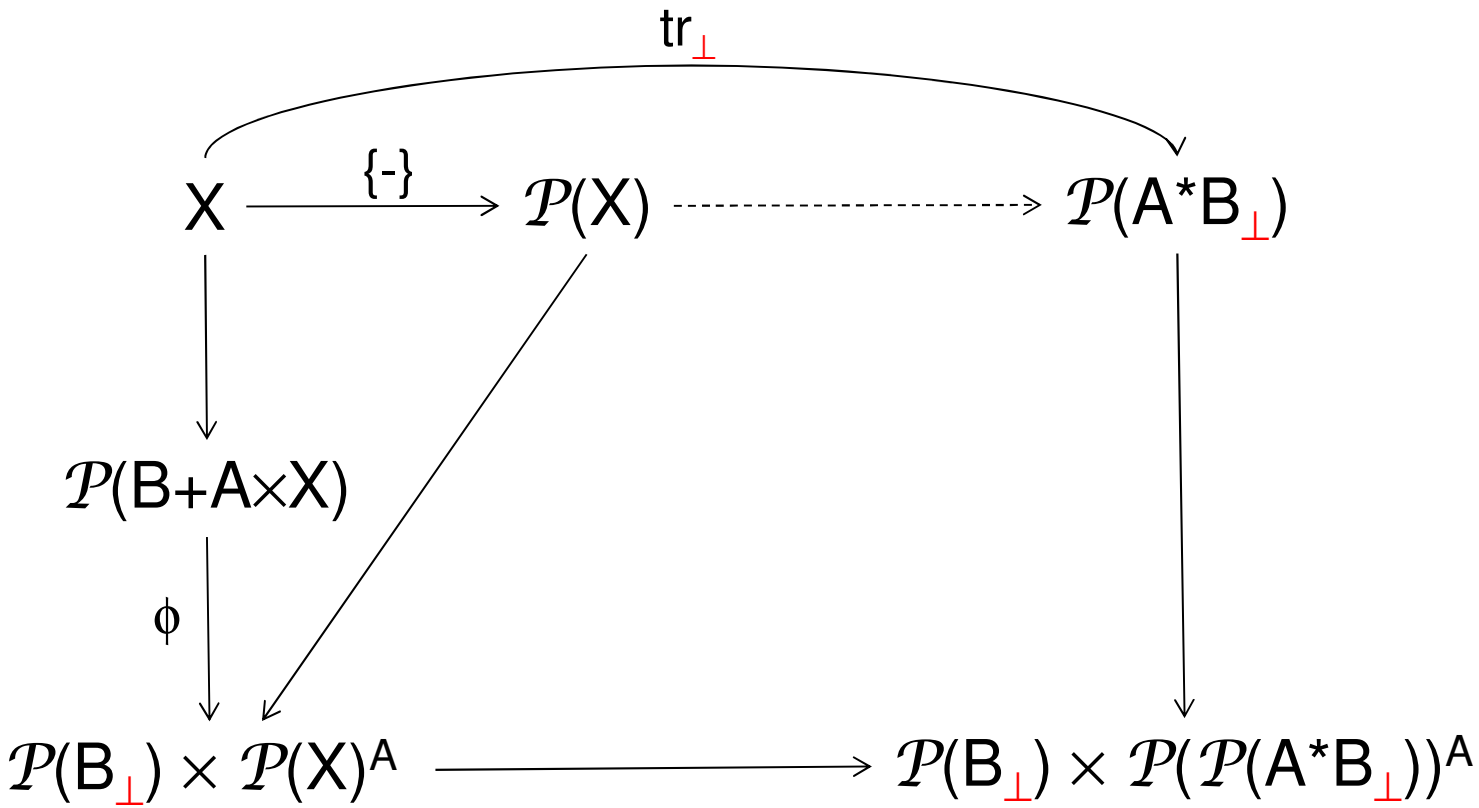


y = <a, y>



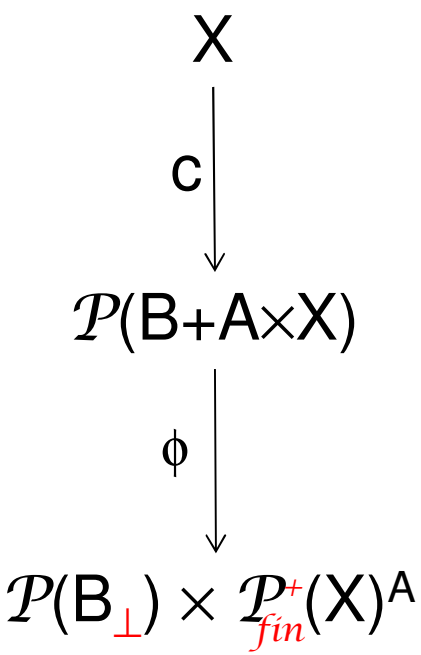
$\phi(S) = \langle (B \cap S) \cup \{\perp\}, \lambda a.\{x \mid \langle a,x\rangle \in S\}\rangle$

Maximal traces via approximation



$$\phi(S) = \langle (B \cap S) \cup \{\perp\}, \lambda a.\{x \mid \langle a,x \rangle \in S\} \rangle$$

Approximations are sound and precise



If c is image-finite then for all $x,y \in X$

$$tr_\perp(x) \subseteq tr_\perp(y)$$

iff

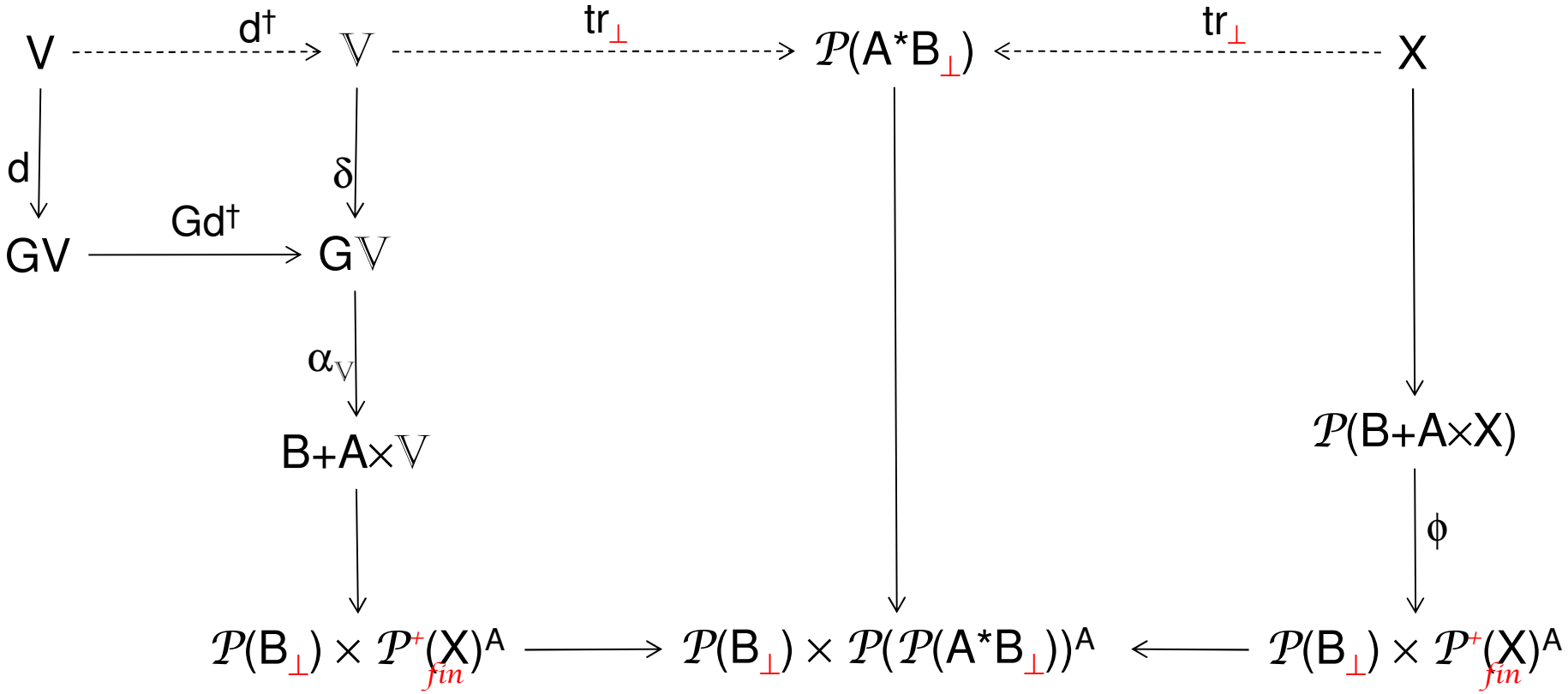
$$T(x) \subseteq T(y)$$

iff

$$x \prec y$$

$$\phi(S) = \langle (B \cap S) \cup \{\perp\}, \lambda a.\{x \mid \langle a,x \rangle \in S\} \rangle$$

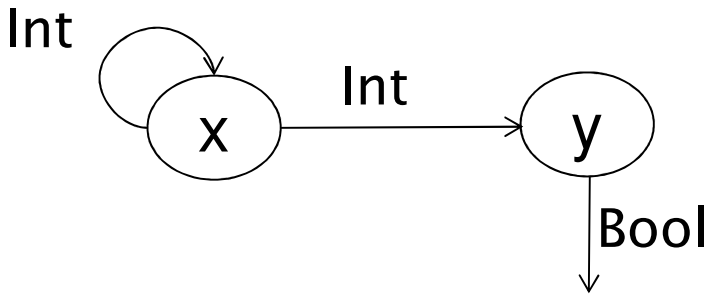
Type assignment



Type assignment v:x iff $tr_\perp(d^\dagger(v)) \subseteq tr_\perp(x)$

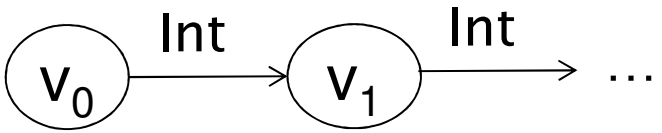
Type assignment

Type



Value

$v_n = \langle n, v_{n+1} \rangle$

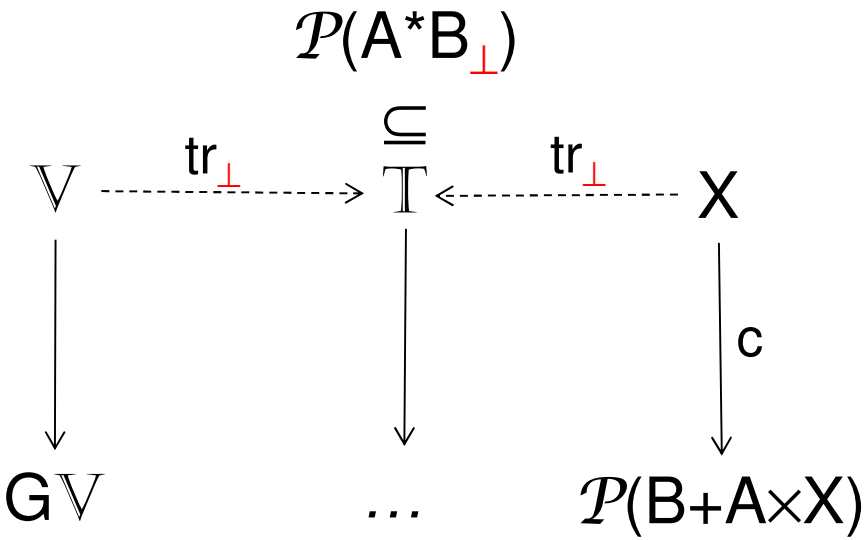


For all n, v_n:x

Types as set of values

$[\![-]\!] : \mathcal{P}(A^*B_\perp) \to \mathcal{P}(\mathbb{V})$

$[\![t]\!] = \{v \in \mathbb{V} \mid tr_\perp(v) \subseteq t\}$



Soundness and completeness

If α is pointwise surjective and c is image finite then for all $x,y \in X$

$$[\![tr_\perp(x)]\!] \subseteq [\![tr_\perp(y)]\!]$$

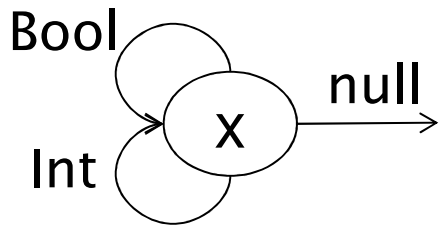
iff

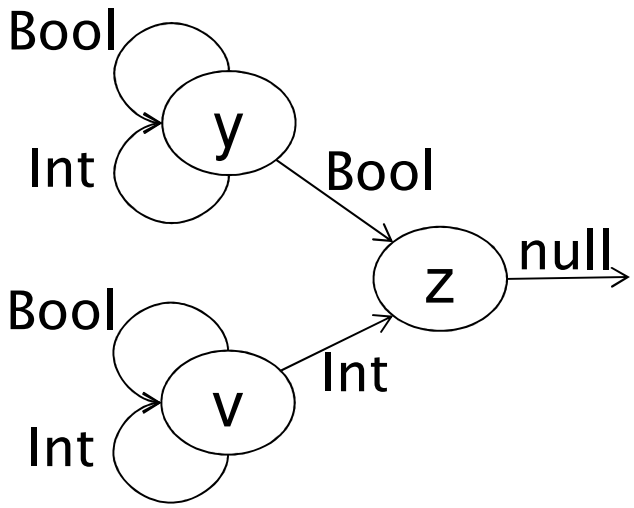
$$tr_\perp(x) \subseteq tr_\perp(y)$$

iff

$$T(x) \subseteq T(y)$$

Example, revised





({x},{y,v,z}) is a bisimulation on the determinization + approximation of the two systems above, thus

$$x \text{ is a subtype of } y \vee v \vee z$$

(and viceversa).

Roadmap

- Plain coinductive types
- Coinductive union types for lists
- Coinductive union types

Non-deterministic top-down tree automata

- $\Sigma X = \coprod_i X^{n_i}$
- Final semantics of Σ is the set of finite and infinite Σ-trees

- Coinductive union types are coalgebras $X \to \mathcal{P}(\Sigma X)$, i.e. non-deterministic top-down tree automata

- Final semantics in Rel, instead of Set, is finite Σ-tree languages.

- The rest is as before …

Conclusions

- A general theory of coinductive union types
- Algorithmically interesting: (bi)simulation, (tree) language containment.
- Semantic based specification method for coalgebras

Further ideas

- Axiomatization of subtype relation
- Generalization to finitary weak-pullbacks preserving functors