Blank Space

Co-implication and its uses in Algebraic Type Theory

Hewitt, Timothy

UTDREX 2017

Purpose

- Introduce Algebraic Type Theory via Category Theory
- Explain the Why behind Co-Implication
- Show fully dual type theory

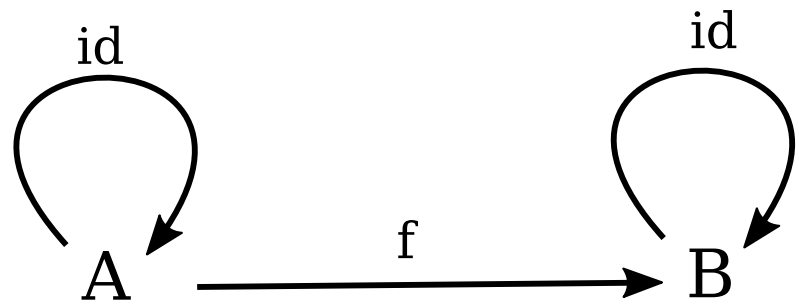
Type Theory

- Alternate to Set theory in mathematics
- Also useful for formal analysis of programming
- Algebraic Type Theory has particular constructors

Category Theory

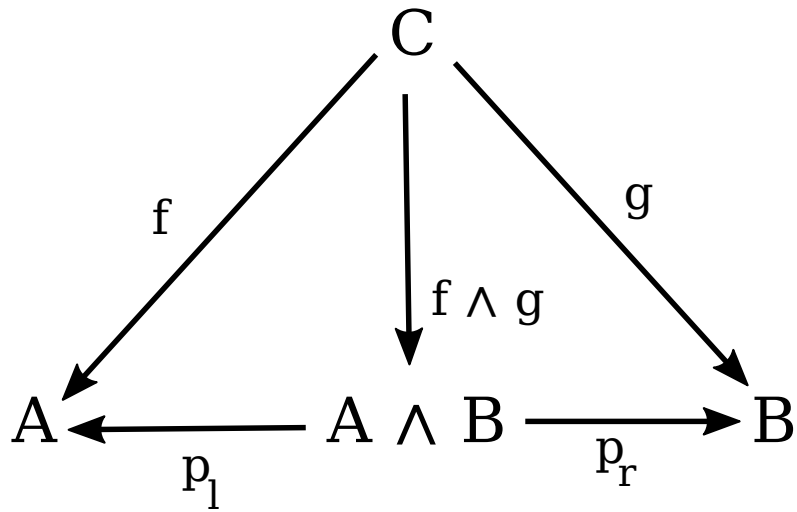
Categories are made of:

- Objects
- Arrows
- Arrows compose, objects have identity



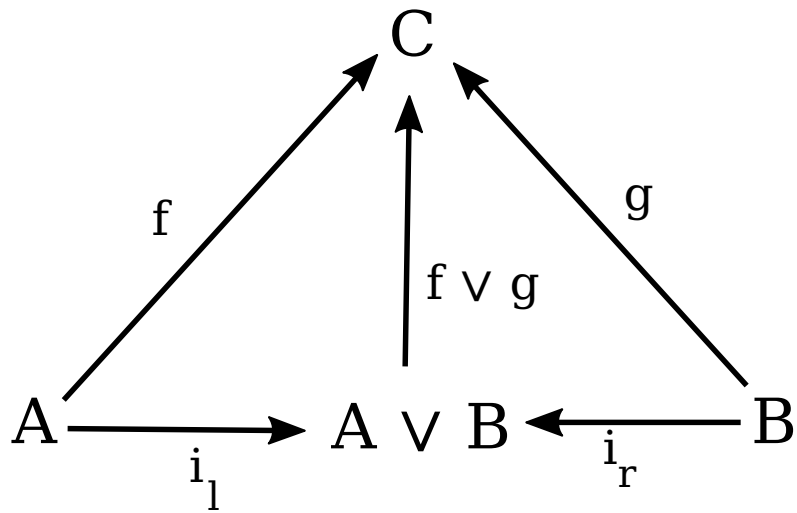
Struct

- Formally known as Pair
- Can be deconstructed into either constituent type
- Combines functions to handle both parts

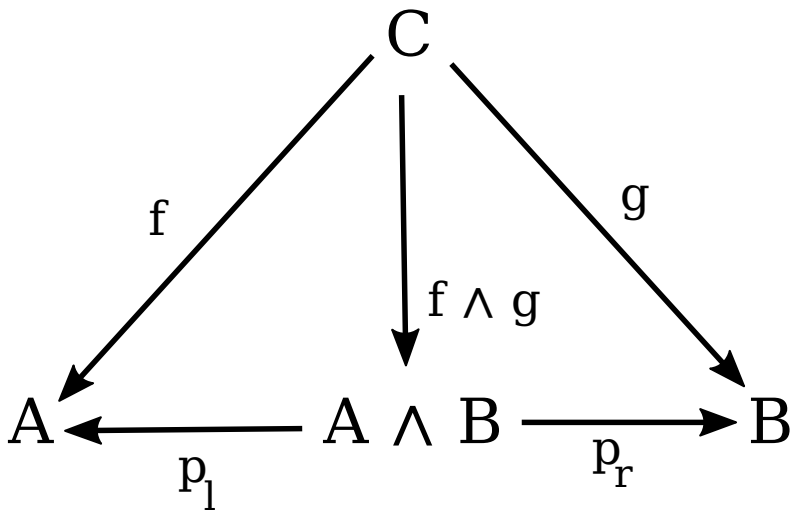


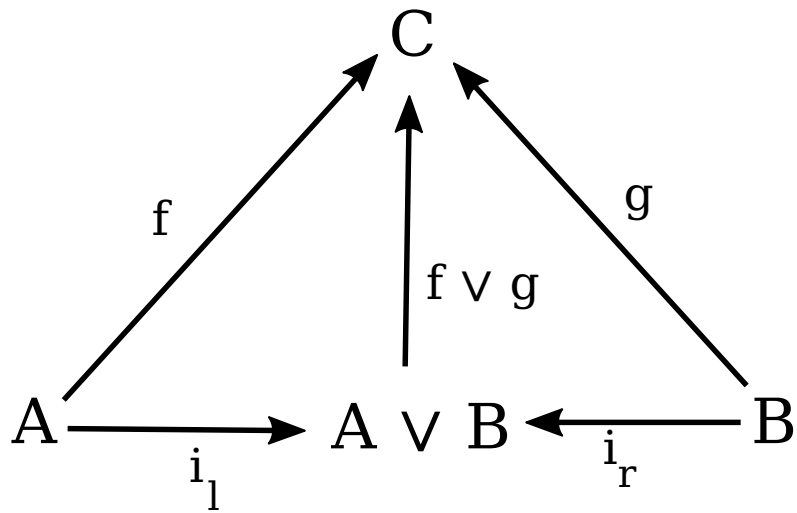
Union

- Tagged union, actually useful
- Can be constructed from either constituent type
- Combines functions to handle either case



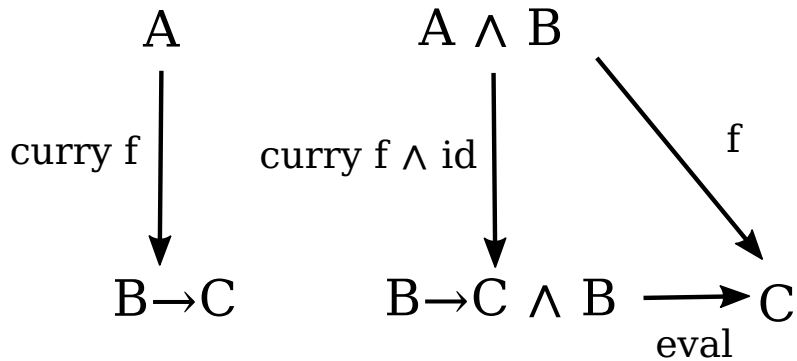
Duality





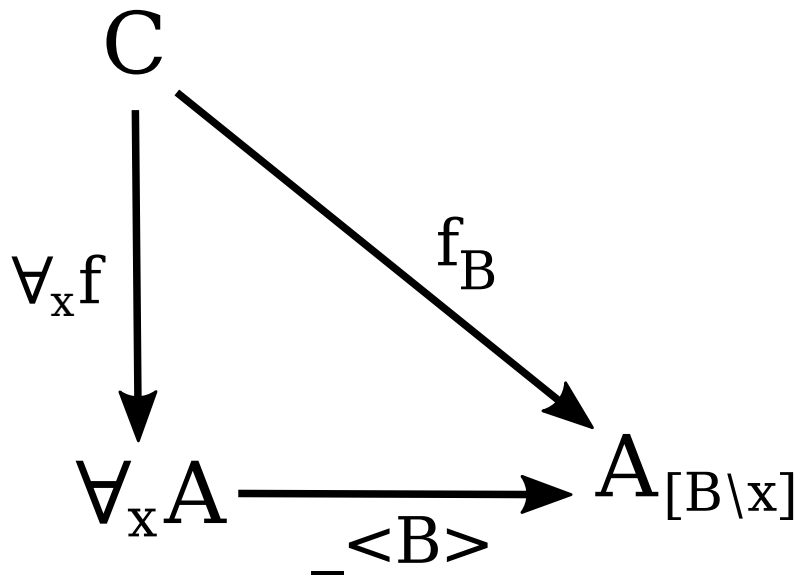
Function

- Pure function, also represents implication in logic
- Maps values to values deterministically
- Combines functions to create functions (triadic composition)



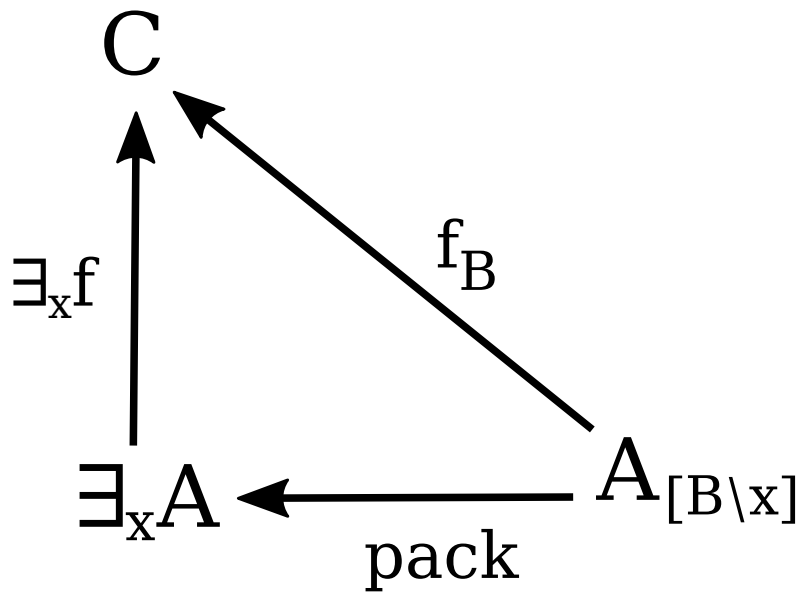
Generic

- Parametric polymorphism
- Maps types to new types
- Introduces type variables to function definitions

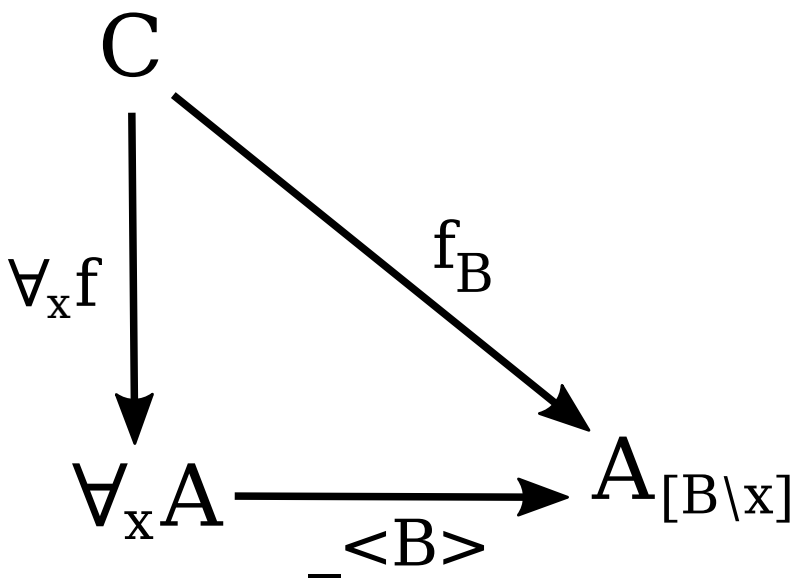


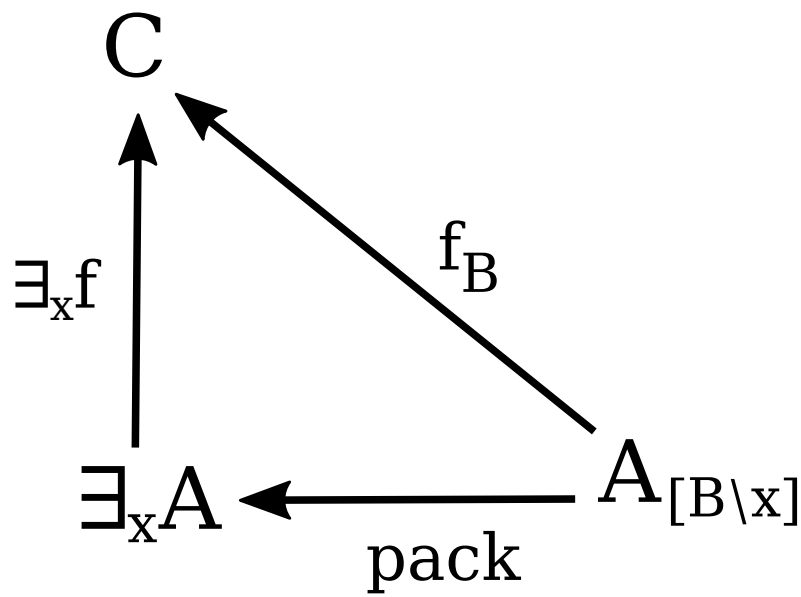
Interface

- Inclusion polymorphism
- Constructed from satisfactory collection of terms
- Restricts functions to only use the guarenteed terms



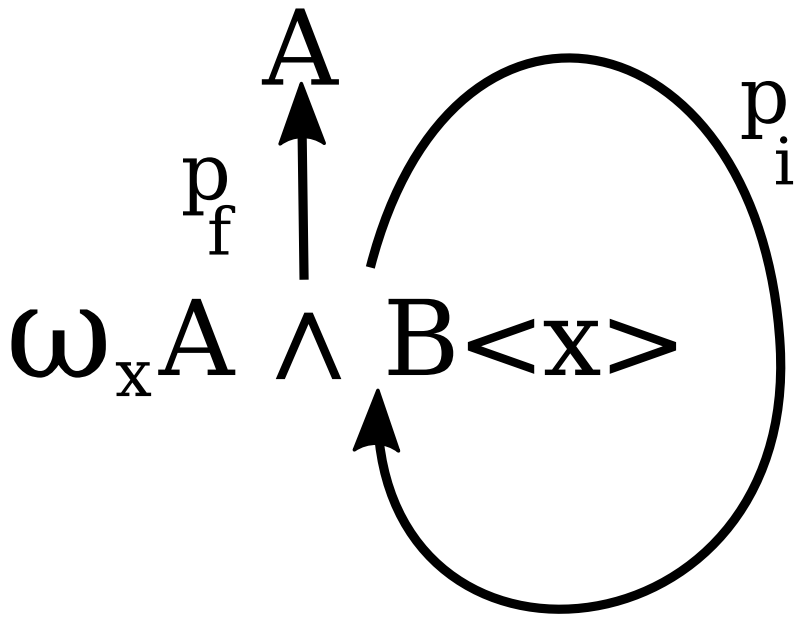
Duality


C
$\forall_x f$
f_B
$\forall_x A$
$A_{[B\backslash x]}$
_<B>


C
$\exists_x f$
f_B
$\exists_x A$
$A_{[B\backslash x]}$
pack

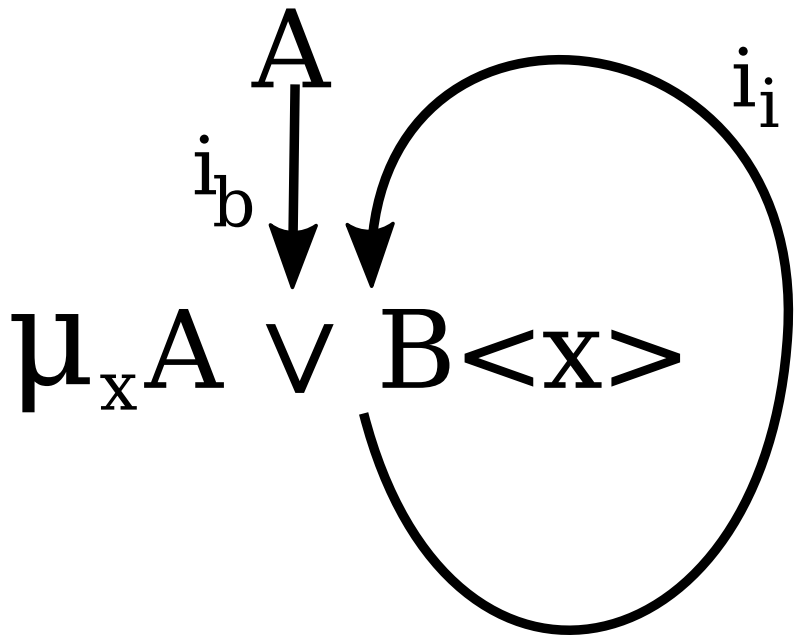
Coalgebra

- Indefinite structures, streams, circular buffers, etc.
- Deconstructed into finite piece and indefinite remainder
- Constructs lazy recursive functions

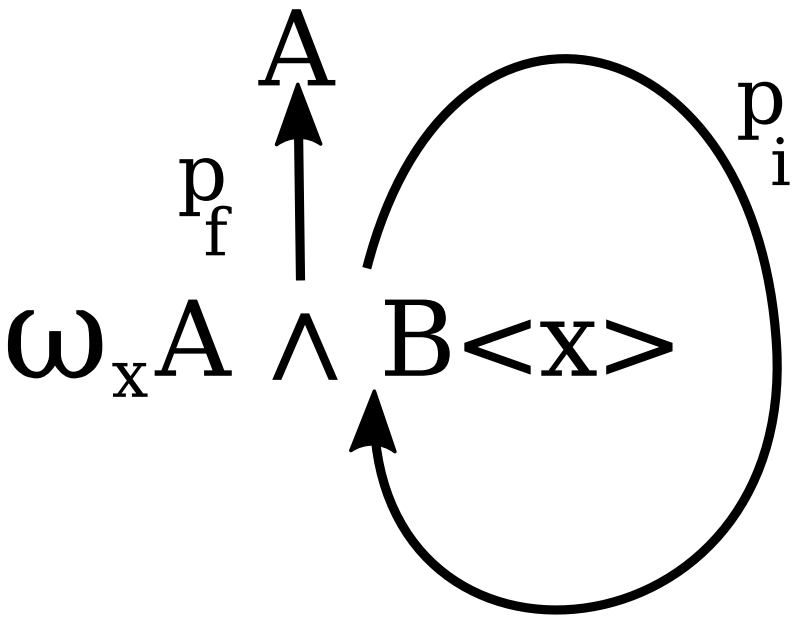


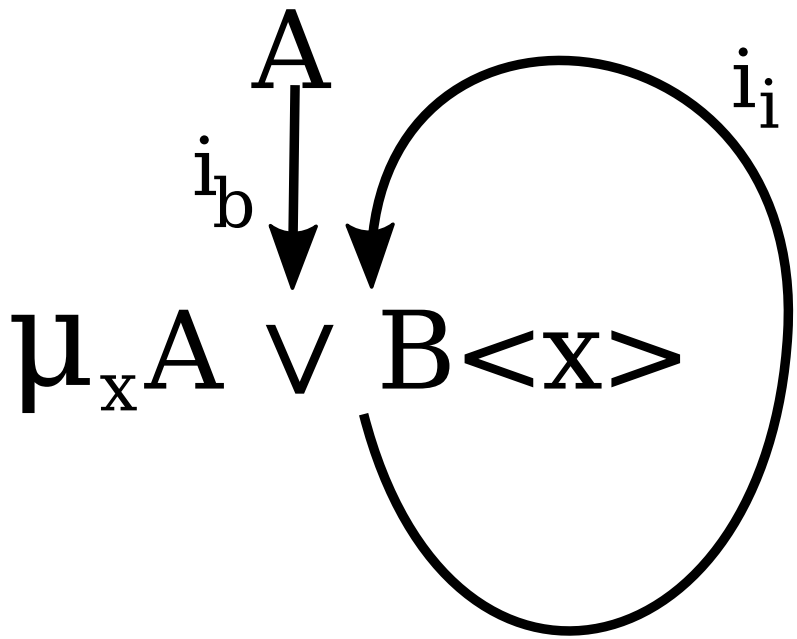
Algebra

- Finite repeating structures, lists, naturals, etc.
- Constructed from finite piece and smaller structure
- Constructs eager recursive functions



Duality



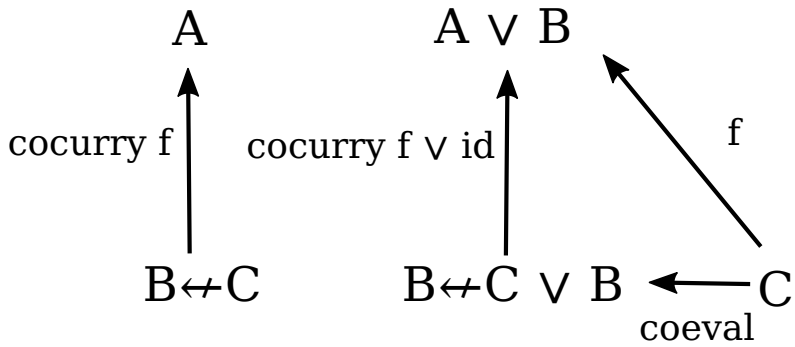


Something seems to be missing

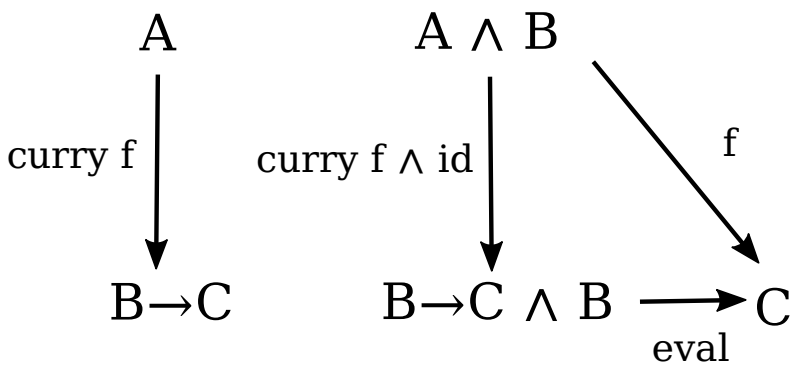
- Every type constructor except Function has a dual
- Function and Generic are similar
- Function's dual should look like Interface

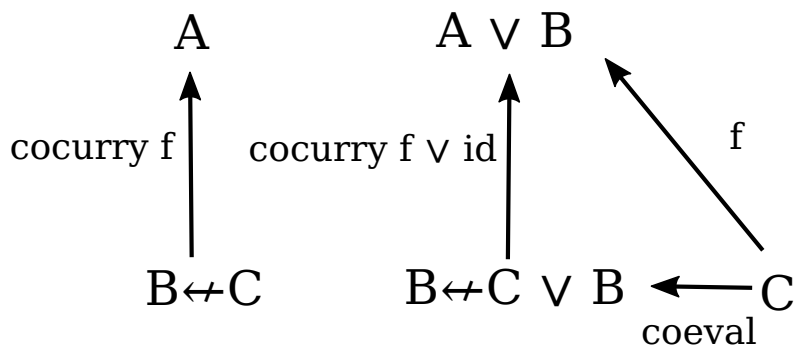
Co-implication

- Referential type, represents converse nonimplication
- Constructed from right parameter that cannot map to left
- Lifts functions to work on references

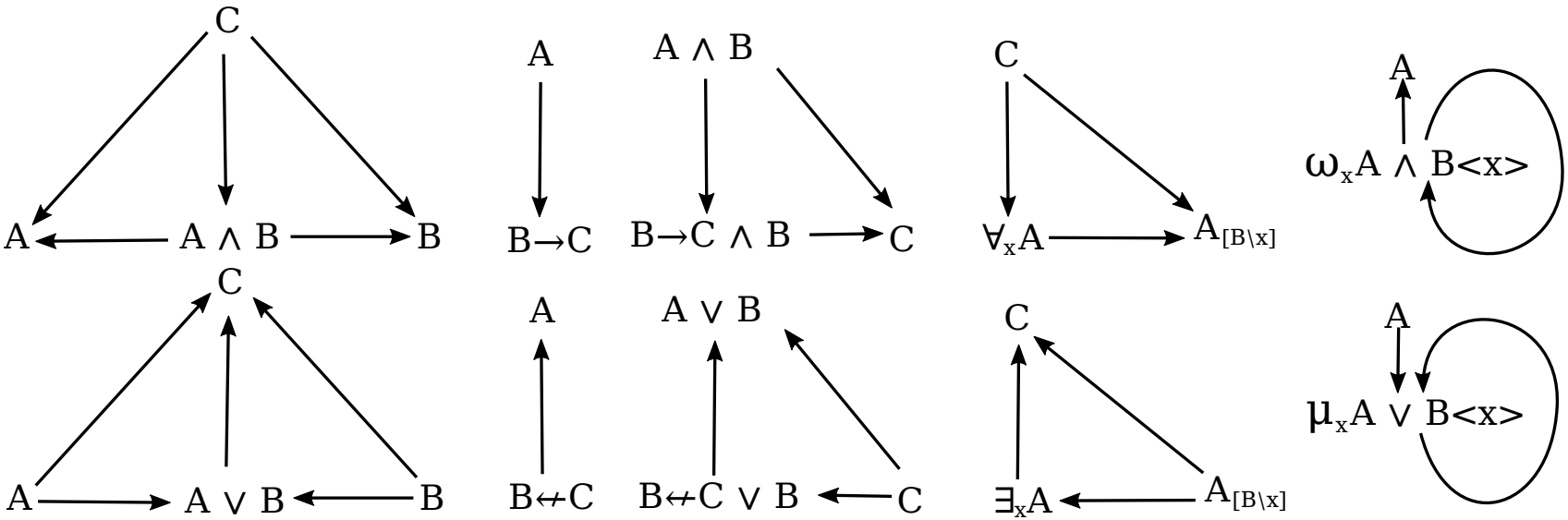


Duality


A
curry f
B→C
A ∧ B
curry f ∧ id
f
B→C ∧ B
eval
C


A
cocurry f
B↚C
A ∨ B
cocurry f ∨ id
f
B↚C ∨ B
coeval
C

Full Duality



So What?

- Encodes nearly any non-dependent type
- Yet to be implemented in entirety
- Stepping stone to fully automated programming

Blank Space

Co-implication and its uses in Algebraic Type Theory

Hewitt, Timothy

UTDREX 2017