

Structures for Structural Recursion

Paul Downen Philip Johnson-Freyd
Zena M. Ariola

University of Oregon

ICFP'15, August 31 – September 2, 2015

Induction and Co-induction

Well-founded recursion

- ▶ Well-foundedness implies termination of some sort
- ▶ *No infinite loops*
- ▶ Two dual flavors: induction and co-induction

Induction

data Nat **where** **data** List *a* **where**
Z : Nat Nil : List *a*
S : Nat → Nat Cons : *a* → List *a* → List *a*

length : $\forall a. \text{List } a \rightarrow \text{Nat}$

length Nil = Z

length (Cons *x* *xs*) = **let** *y* = *length* *xs* **in** S *y*

Co-induction

codata InfList *a* **where**

Cons : *a* → InfList *a* → InfList *a*

zeroes : InfList Nat

zeroes = *Cons* Z *zeroes*

count : Nat → InfList Nat

count *x* = *Cons* *x* (*count* *S*(*x*))

Co-induction

codata Stream a **where**

Head : Stream $a \rightarrow a$

Tail : Stream $a \rightarrow$ Stream a

$zeroes$: Stream Nat

$zeroes$.Head = Z

$zeroes$.Tail = $zeroes$

$count$: Nat \rightarrow Stream Nat

$(count\ x)$.Head = x

$(count\ x)$.Tail = $count\ (x + 1)$

Well-founded induction and co-induction

- ▶ Well-foundedness for induction is clear
 - ▶ Structural induction
- ▶ Well-foundedness for co-induction is murky
 - ▶ Productivity? Guardedness?
- ▶ Asymmetric bias for induction over co-induction
- ▶ Can they be unified?
- ▶ Idea: Complete *symmetry* to find *structure*

Recursion on Structures

Classical sequent calculus: a symmetric language

- ▶ Producers (terms):

$$v \in \mathit{Term} ::= x \mid \mu\alpha.c \mid \dots$$

- ▶ Consumers (co-terms):

$$e \in \mathit{CoTerm} ::= \alpha \mid \tilde{\mu}x.c \mid \dots$$

- ▶ Computations (commands):

$$c \in \mathit{Command} ::= \langle v \parallel e \rangle$$

Input and output

A place for everything and everything in its place.

- ▶ Computations do not return, they *run*
- ▶ Unspecified inputs (x, y, z) and outputs (α, β, γ)
- ▶ $\tilde{\mu}$ abstracts over unspecified input

$$\langle x \parallel \tilde{\mu} y . c \rangle = c\{y/x\}$$

- ▶ μ abstracts over unspecified output

$$\langle \mu \beta . c \parallel \alpha \rangle = c\{\beta/\alpha\}$$

Data types

- ▶ Values are *constructed*
- ▶ Consumed by *pattern matching*

data Nat where

Z : \vdash Nat |

S : Nat \vdash Nat |

data List(a) where

Nil : \vdash List(a) |

Cons : a, List(a) \vdash List(a) |

Co-data types

- ▶ Observations are *constructed*
- ▶ Produced by *pattern matching*

codata $a \rightarrow b$ **where**

$-\cdot- : a \mid a \rightarrow b \vdash b$

codata $\text{Stream}(a)$ **where**

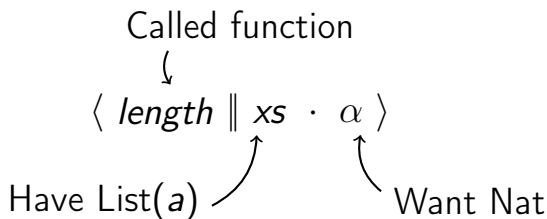
Head : $\mid \text{Stream}(a) \vdash a$

Tail : $\mid \text{Stream}(a) \vdash \text{Stream}(a)$

User-defined (co-)data types

- ▶ All types user-definable, follow same pattern
- ▶ ADTs from functional languages are data
- ▶ Functions are co-data
- ▶ Universal quantification is co-data
 - ▶ Explicit \forall à la System F_ω
- ▶ Existential quantification is data
- ▶ Types that lie *outside* the functional paradigm

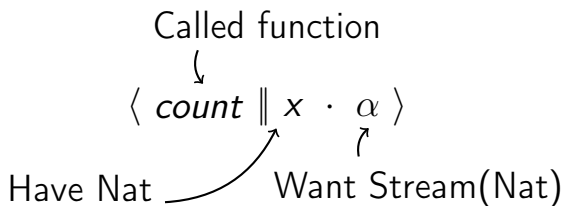
Recursion on data structures



$$\langle \textit{length} \parallel \text{Nil} \cdot \alpha \rangle = \langle Z \parallel \alpha \rangle$$

$$\langle \textit{length} \parallel \text{Cons}(x, xs) \cdot \alpha \rangle = \langle \textit{length} \parallel xs \cdot \tilde{\mu}y. \langle S(y) \parallel \alpha \rangle \rangle$$

Recursion on co-data structures



$$\langle \textit{count} \parallel x \cdot \text{Head}[\alpha] \rangle = \langle x \parallel \alpha \rangle$$

$$\langle \textit{count} \parallel x \cdot \text{Tail}[\alpha] \rangle = \langle \textit{count} \parallel S(x) \cdot \alpha \rangle$$

Structural recursion

- ▶ Distinction between induction and co-induction fade away
- ▶ Both are modes of recursion on *some* structure
 - ▶ Induction: recurse on data structure value
 - ▶ Co-induction: recurse on co-data structure observation
- ▶ Recursive invocations run with sub-structures

$$\langle \text{length} \parallel \text{Cons}(x, xs) \cdot \alpha \rangle = \langle \text{length} \parallel xs \cdot \tilde{\mu}y. \langle S(y) \parallel \alpha \rangle \rangle$$
$$\langle \text{count} \parallel x \cdot \text{Tail}[\alpha] \rangle = \langle \text{count} \parallel S(x) \cdot \alpha \rangle$$

Structures for Recursion

Finding the sub-structure

- ▶ To check well-foundedness, check for decreasing sub-structure
- ▶ But relevant sub-structure appears inside a larger structural context

$$\begin{aligned}\langle \text{length} \parallel \text{Cons}(x, xs) \cdot \alpha \rangle &= \langle \text{length} \parallel xs \cdot \tilde{\mu}y. \langle \text{S}(y) \parallel \alpha \rangle \rangle \\ \langle \text{count} \parallel x \cdot \text{Tail}[\alpha] \rangle &= \langle \text{count} \parallel \text{S}(x) \cdot \alpha \rangle\end{aligned}$$

- ▶ Structure of function calls not special, same for tuples, etc.
- ▶ How do we know where to find it?

Tracking sub-structures with sized types

- ▶ Type-based approach to termination
- ▶ Size approximate the depth of structures
- ▶ Types can be indexed by (several) sizes
- ▶ Separate recursion in types from recursion in programs

Recursion in types

- ▶ Add extra size index to recursive (co-)data types
- ▶ Change in size tracks recursive sub-structures of recursive types
- ▶ Given $x : \text{Nat}(i)$ then $S(x) : \text{Nat}(i + 1)$
- ▶ Given $\alpha : \text{Stream}(i, a)$ then
 $\text{Tail}[\alpha] : \text{Stream}(i + 1, a)$

Recursion in programs

- ▶ Recursion over structures of recursive type *quantifies* over size index
- ▶ $length : \forall a. \forall i. List(i, a) \rightarrow Nat(i)$
- ▶ $count : \forall i. (\exists j. Nat(j)) \rightarrow Stream(i, \exists j. Nat(j))$
- ▶ Different kinds of sizes for different purposes:
 - ▶ Step-by-step (primitive) recursion: computation *depends* on type-level size index at run-time, dependently typed vectors
 - ▶ Bounded (noetherian) recursion: type-level size index is *erasable* at run-time, recurse on deeply nested sub-structure

Structures for structural recursion

- ▶ Size quantifiers *are* themselves (co-)data types
- ▶ Their values and observations are structures for specifying structural recursion
- ▶ Like \forall and \exists , quantify sizes over arbitrary types
- ▶ Can “induct” over co-data types, vice versa
 - ▶ Eliminate the need for strictures on structures

More in the paper

- ▶ Source effect-free functional calculus with recursion, data types, and “pure” objects
- ▶ Target classical calculus with user-defined recursive (co-)data and recursion schemes
- ▶ Modest dependent types with control effects
- ▶ Different evaluation strategies, parametrically
- ▶ Strong normalization
- ▶ Type erasure and computationally relevant types

Final thoughts

- ▶ Induction and co-induction are modes of structural recursion
- ▶ Find the structure with both sides of the story
- ▶ Duality and symmetry are powerful weapons: they invert murky problems into clear ones