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

$$\begin{array}{ll}
\textbf{data}\ \mathsf{Nat}\ \textbf{where} & \textbf{data}\ \mathsf{List}\ a\ \textbf{where} \\
\quad \mathsf{Z} : \mathsf{Nat} & \quad \mathsf{Nil} : \mathsf{List}\ a \\
\quad \mathsf{S} : \mathsf{Nat} \to \mathsf{Nat} & \quad \mathsf{Cons} : a \to \mathsf{List}\ a \to \mathsf{List}\ a
\end{array}$$

$$\begin{array}{ll}
\mathit{length} & : \forall a.\ \mathsf{List}\ a \to \mathsf{Nat} \\
\mathit{length}\ \ \mathsf{Nil} & = \mathsf{Z} \\
\mathit{length}\ (\mathsf{Cons}\ x\ \ xs) & = \textbf{let}\ y = \mathit{length}\ xs\ \textbf{in}\ \mathsf{S}\ y
\end{array}$$

Co-induction

$$\textbf{codata}\ \mathsf{InfList}\ a\ \textbf{where}$$
$$\quad \mathsf{Cons} : a \to \mathsf{InfList}\ a \to \mathsf{InfList}\ a$$

$$\begin{aligned} \mathit{zeroes} &: \mathsf{InfList}\ \mathsf{Nat} \\ \mathit{zeroes} &= \mathsf{Cons}\ \mathsf{Z}\ \ \mathit{zeroes} \end{aligned}$$

$$\begin{aligned} \mathit{count} \quad &: \mathsf{Nat} \to \mathsf{InfList}\ \mathsf{Nat} \\ \mathit{count}\ x &= \mathsf{Cons}\ x\ (\mathit{count}\ \ \mathsf{S}(x)) \end{aligned}$$

Co-induction

$$\textbf{codata}\ \mathrm{Stream}\ a\ \textbf{where}$$

$$\begin{aligned}\mathrm{Head} &: \mathrm{Stream}\ a \to a\\ \mathrm{Tail} &: \mathrm{Stream}\ a \to \mathrm{Stream}\ a\end{aligned}$$

$$\begin{aligned}&zeroes && : \mathrm{Stream}\ \mathrm{Nat}\\ &zeroes.\mathrm{Head} && = \mathrm{Z}\\ &zeroes.\mathrm{Tail} && = zeroes\end{aligned}$$

$$\begin{aligned}&count && : \mathrm{Nat} \to \mathrm{Stream}\ \mathrm{Nat}\\ &(count\ x).\mathrm{Head} && = x\\ &(count\ x).\mathrm{Tail} && = count\ (x+1)\end{aligned}$$

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):

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

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

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

- μ abstracts over unspecified output

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

Data types

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

$$
\begin{array}{lrl}
\textbf{data}\,\mathsf{Nat}\,\textbf{where} & & \\
\quad \mathsf{Z} : & \vdash \mathsf{Nat} & | \\
\quad \mathsf{S} : & \mathsf{Nat} \vdash \mathsf{Nat} & |
\end{array}
\qquad
\begin{array}{lrl}
\textbf{data}\,\mathsf{List}(a)\,\textbf{where} & & \\
\quad \mathsf{Nil} : & \vdash \mathsf{List}(a) & | \\
\quad \mathsf{Cons} : & a, \mathsf{List}(a) \vdash \mathsf{List}(a) & |
\end{array}
$$

Co-data types

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

$$\textbf{codata}\; a \to b \;\textbf{where}$$
$$\_ \cdot \_ : \quad a \mid a \to b \vdash b$$

$$\textbf{codata}\; \mathsf{Stream}(a) \;\textbf{where}$$
$$\mathsf{Head} : \quad \mid \mathsf{Stream}(a) \vdash a$$
$$\mathsf{Tail} : \quad \mid \mathsf{Stream}(a) \vdash \mathsf{Stream}(a)$$

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

Called function

$$\langle\ \mathit{length}\ \|\ \mathit{xs}\ \cdot\ \alpha\ \rangle$$

Have List(a)

Want Nat

$$
\begin{aligned}
\langle \mathit{length} \| \mathsf{Nil} \cdot \alpha \rangle &= \langle \mathsf{Z} \| \alpha \rangle \\
\langle \mathit{length} \| \mathsf{Cons}(x, \mathit{xs}) \cdot \alpha \rangle &= \langle \mathit{length} \| \mathit{xs} \cdot \tilde{\mu} y. \langle \mathsf{S}(y) \| \alpha \rangle \rangle
\end{aligned}
$$

Recursion on co-data structures

Called function

$$\langle \; \mathit{count} \parallel x \;\cdot\; \alpha \; \rangle$$

Have Nat

Want Stream(Nat)

$$\begin{aligned}
\langle \mathit{count} \| x \cdot \mathrm{Head}[\alpha] \rangle &= \langle x \| \alpha \rangle \\
\langle \mathit{count} \| x \cdot \mathrm{Tail}[\alpha] \rangle &= \langle \mathit{count} \| \mathrm{S}(x) \cdot \alpha \rangle
\end{aligned}$$

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

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

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

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

- 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 $x : \mathsf{Nat}(i)$ then $\mathsf{S}(x) : \mathsf{Nat}(i + 1)$
- Given $\alpha : \mathsf{Stream}(i, a)$ then $\mathsf{Tail}[\alpha] : \mathsf{Stream}(i + 1, a)$

Recursion in programs

- Recursion over structures of recursive type *quantifies* over size index
- *length* : $\forall a. \forall i. \, \mathsf{List}(i, a) \rightarrow \mathsf{Nat}(i)$
- *count* : $\forall i. (\exists j. \, \mathsf{Nat}(j)) \rightarrow \mathsf{Stream}(i, \exists j. \, \mathsf{Nat}(j))$
- 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