Mixing Induction and Coinduction

Nils Anders Danielsson (Nottingham)

Aussois, 2009-05-15

Introduction

- Purely coinductive definitions are sometimes too “coarse”.
- Mixing in a bit of induction gives more precision.
- This technique does not seem to be well-known.
- I will introduce it through some examples.

Coinductive types in Agda

Streams

$$\mathbf{data}\ \mathit{Stream}\ (A\ :\ \mathit{Set})\ :\ \mathit{Set}\ \mathbf{where}$$
$$\quad \_{::}\_\ :\ A\ \rightarrow\ \infty\,(\mathit{Stream}\ A)\ \rightarrow\ \mathit{Stream}\ A$$

∞

- ∞ marks coinductive arguments.
- $T \; = \; F \, (\infty \; T) \; T \quad \approx \quad T \; = \; \nu C . \mu I . \; F \; C \; I.$
- Can be seen as a suspension.
- Delay and force:

$$\begin{array}{l} \sharp \; : \; \forall \, \{A\} \; \rightarrow \; A \; \rightarrow \; \infty \; A \\ \flat \; : \; \forall \, \{A\} \; \rightarrow \; \infty \; A \; \rightarrow \; A \end{array}$$

∞

- ∞ marks coinductive arguments.
- $T = F\,(\infty\; T)\; T \quad \approx \quad T = \nu C.\mu I.\; F\; C\; I.$
- Can be seen as a suspension.
- Delay and force:

$$\textbf{codata}\ \infty\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}$$
$$\sharp : A \to \infty\ A$$

$$\flat : \forall\ \{A\} \to \infty\ A \to A$$
$$\flat\ (\sharp\ x) = x$$

Streams and stream processors

$Stream\ A \ \approx\ \nu C . A \times C$:

$$
\begin{aligned}
&\textbf{data}\ Stream\ (A : Set) : Set\ \textbf{where}\\
&\quad \_{::}\_ : A \to \infty\,(Stream\ A) \to Stream\ A
\end{aligned}
$$

$SP\ A\ B \ \approx\ \nu C . \mu I .\ (A \to I) + B \times C$:

$$
\begin{aligned}
&\textbf{data}\ SP\ (A\ B : Set) : Set\ \textbf{where}\\
&\quad \mathrm{get} : (A \to SP\ A\ B) \to SP\ A\ B\\
&\quad \mathrm{put} : B \to \infty\,(SP\ A\ B) \to SP\ A\ B
\end{aligned}
$$

Guarded corecursion

$$
\begin{aligned}
&\mathit{map} : \forall \{A\ B\} \to (A \to B) \to \mathit{Stream}\ A \to \mathit{Stream}\ B \\
&\mathit{map}\ f\ (x :: xs) = f\ x :: \sharp\ (\mathit{map}\ f\ (\flat\ xs))
\end{aligned}
$$

Lexicographic guarded corecursion and
structural recursion:

$$
\begin{aligned}
&[\![\_]\!] : \forall \{A\ B\} \to \mathit{SP}\ A\ B \to \mathit{Stream}\ A \to \mathit{Stream}\ B \\
&[\![\ \mathrm{get}\ f\ ]\!]\ (a :: as) = [\![\ f\ a\ ]\!]\ (\flat\ as) \\
&[\![\ \mathrm{put}\ b\ sp\ ]\!]\ as = b :: \sharp\ ([\![\ \flat\ sp\ ]\!]\ as)
\end{aligned}
$$

"Coinductive families"

$$
\begin{array}{l}
\textbf{data}\ \_\approx\_\ \{A\}\ :\ \mathit{Stream}\ A \to \mathit{Stream}\ A \to \mathit{Set}\ \textbf{where} \\
\quad \_::\_\ :\ \forall\ x\ \{\mathit{xs}\ \mathit{ys}\} \to \infty\ (\flat\ \mathit{xs}\ \approx\ \flat\ \mathit{ys}) \to \\
\qquad\qquad x :: \mathit{xs}\ \approx\ x :: \mathit{ys}
\end{array}
$$

Guarded coinduction:

$$
\begin{array}{l}
\mathit{map\text{-}cong}\ :\ \forall\ \{A\ B\}\ (f\ :\ A \to B)\ \{\mathit{xs}\ \mathit{ys}\} \to \\
\qquad\qquad\qquad \mathit{xs}\ \approx\ \mathit{ys} \to \mathit{map}\ f\ \mathit{xs}\ \approx\ \mathit{map}\ f\ \mathit{ys} \\
\mathit{map\text{-}cong}\ f\ (x :: \mathit{xs}{\approx}\mathit{ys})\ = \\
\quad f\ x :: \sharp\ (\mathit{map\text{-}cong}\ f\ (\flat\ \mathit{xs}{\approx}\mathit{ys}))
\end{array}
$$

Inference systems

Inference systems

- Two kinds of inference systems:
  - Algorithmic (syntax-directed).
  - Declarative (with rules like transitivity).
- Declarative coinductive inference systems are often a bad idea:

$$
\begin{aligned}
bad &: \forall \{x\ y\} \rightarrow x \approx y \\
bad &= \mathsf{trans}\ (\sharp\ bad)\ (\sharp\ bad)
\end{aligned}
$$

- Solution: Make non-structural rules inductive.

Alternative definition of stream equality

$$
\begin{array}{lll}
\textbf{data}\ \_\sim\_\ \{A\} & : & \mathit{Stream}\ A \to \mathit{Stream}\ A \to \mathit{Set}\ \textbf{where} \\
\quad \_::\_ & : & \forall\ x\ \{xs\ ys\} \to \infty\ (\flat\ xs\ \sim\ \flat\ ys) \to \\
 & & x :: xs\ \sim\ x :: ys \\
\quad \mathsf{refl} & : & \forall\ \{xs\} \to xs\ \sim\ xs \\
\quad \mathsf{sym} & : & \forall\ \{xs\ ys\} \to xs\ \sim\ ys \to ys\ \sim\ xs \\
\quad \mathsf{trans} & : & \forall\ \{xs\ ys\ zs\} \to \\
 & & xs\ \sim\ ys \to ys\ \sim\ zs \to xs\ \sim\ zs
\end{array}
$$

Equivalent to $\_\approx\_$.

Parser combinators

Parser combinators

- Parser combinators are nice.
- But what about termination?
- Left recursion often problematic:

$$\begin{array}{rcl} \mathit{expr} & = & \mathit{expr} \cdot \mathit{tok}\ \texttt{"+"} \cdot \mathit{term} \\ & \mid & \mathit{term} \\ \mathit{term} & = & \ldots \end{array}$$

Interface (roughly)

$$
\begin{array}{lcl}
G & : & Set \\
\_\in\_ & : & List\ Token \ \rightarrow \ G \ \rightarrow \ Set \\
\_\in?\_ & : & \forall\ s\ g \ \rightarrow \ Dec\ (s \in g)
\end{array}
$$

Note that $\_\in?\_$ returns an inductive result.

Interface (roughly)

$$
\begin{array}{lcl}
\emptyset & : & G \\
\varepsilon & : & G \\
\mathsf{tok} & : & \mathit{Token} \rightarrow G \\
\_|\_ & : & G \rightarrow G \rightarrow G \\
\_\cdot\_ & : & G \rightarrow G \rightarrow G
\end{array}
$$

Corecursion will be used $\Rightarrow$
some arguments have to be coinductive.

Choice

Hard to decide infinite choice:

$$g = g \mid g'$$
$$g = g' \mid g$$

The arguments of $\_\mid\_$ will be inductive.

Sequencing

Problematic if g′ is nullable, otherwise OK:

$$
\begin{aligned}
g &= g \cdot g' \\
g &= g' \cdot g
\end{aligned}
$$

Let us index G on whether or not
the empty string is accepted.

Conditional coinduction

$$
\begin{aligned}
&\infty? \;:\; Bool \;\to\; Set \;\to\; Set \\
&\infty?\ \text{true}\ A \;=\; \infty\ A \\
&\infty?\ \text{false}\ A \;=\; A
\end{aligned}
$$

$$
\begin{aligned}
&\sharp? \;:\; \forall\ b\ \{A\} \;\to\; A \;\to\; \infty?\ b\ A \\
&\sharp?\ \text{true}\quad x \;=\; \sharp\ x \\
&\sharp?\ \text{false}\quad x \;=\; x
\end{aligned}
$$

$$
\begin{aligned}
&\flat? \;:\; \forall\ b\ \{A\} \;\to\; \infty?\ b\ A \;\to\; A \\
&\flat?\ \text{true}\quad x \;=\; \flat\ x \\
&\flat?\ \text{false}\quad x \;=\; x
\end{aligned}
$$

Grammars

Index true iff empty string accepted:

$$
\begin{array}{lll}
\textbf{data } G & : & Bool \to Set \textbf{ where} \\
\quad \emptyset & : & G \text{ false} \\
\quad \varepsilon & : & G \text{ true} \\
\quad \text{tok} & : & Token \to G \text{ false} \\
\quad \_|\_ & : & \forall \{n_1\ n_2\} \to \\
 & & G\ n_1 \to G\ n_2 \to G\ (n_1 \lor n_2) \\
\quad \_\cdot\_ & : & \forall \{n_1\ n_2\} \to \infty?\ (not\ n_2)\ (G\ n_1) \to \\
 & & \qquad\qquad\quad \infty?\ (not\ n_1)\ (G\ n_2) \to \\
 & & \qquad\qquad\quad G\ (n_2 \land n_1)
\end{array}
$$

Grammars

Index true iff empty string accepted:

$$
\begin{aligned}
&\textbf{data}\ G : \mathit{Bool} \to \mathit{Set}\ \textbf{where}\\
&\quad \emptyset \quad : G\ \mathsf{false}\\
&\quad \varepsilon \quad : G\ \mathsf{true}\\
&\quad \mathsf{tok} : \mathit{Token} \to G\ \mathsf{false}\\
&\quad \_|\_ : \forall\ \{n_1\ n_2\} \to\\
&\qquad\quad G\ n_1 \to G\ n_2 \to G\ (n_1 \lor n_2)\\
&\quad \_\cdot\_ : \forall\ \{n_1\ n_2\} \to G\ n_1 \to\\
&\qquad\qquad\qquad\qquad \infty?\ (\mathit{not}\ n_1)\ (G\ n_2) \to\\
&\qquad\qquad\qquad\qquad G\ (n_1 \land n_2)
\end{aligned}
$$

Example

Kleene star:

mutual

$$
\begin{aligned}
&\_\star \; : \; G \; \text{false} \; \to \; G \; \text{true} \\
&g \star \; = \; \varepsilon \mid g + \\
&\_+ \; : \; G \; \text{false} \; \to \; G \; \text{false} \\
&g + \; = \; g \cdot \sharp \, (g \star)
\end{aligned}
$$

The argument must not accept the empty string;
$\varepsilon \star$ is not very useful.

Semantics

Inductive:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{data}\ \_\in\_ \ :\ \mathit{List}\ \mathit{Token} \to G\ n \to \mathit{Set}\ \textbf{where}} \\
\quad \varepsilon & : & [\,] \in \varepsilon \\
\quad \mathsf{tok} & : & [\,t\,] \in \mathsf{tok}\ t \\
\quad |^{\ell} & : & s \in g_1 \to s \in g_1 \mid g_2 \\
\quad |^{\mathsf{r}} & : & s \in g_2 \to s \in g_1 \mid g_2 \\
\quad \_\cdot\_ & : & s_1 \in g_1 \to s_2 \in \flat ?\ (\mathit{not}\ n_1)\ g_2 \to \\
 & & s_1 +\!\!+\ s_2 \in g_1 \cdot g_2
\end{array}
$$

For $g : G\ n$:

$[\,] \in g$ iff $n \equiv \mathsf{true}$.

Implementation

- Uses a variant of Brzozowski's *Derivatives of Regular Expressions*.
- $\partial : \forall \{n\}\ (g : G\ n)\ (t : Token)\ \to\ G\ (\partial n\ g\ t)$.
- $s \in \partial\ g\ t$ iff $t :: s \in g$.
- ∂ is used once per element in the input string.
- ∂ uses recursion over the inductive structure of the grammars.

∂

$$
\begin{array}{l}
\partial \;:\; \forall \{n\}\,(g \;:\; G\;n)\,(t \;:\; \mathit{Token}) \;\to\; G\,(\partial n\;g\;t) \\
\partial\;\emptyset \qquad\qquad\qquad\qquad t \;=\; \emptyset \\
\partial\;\varepsilon \qquad\qquad\qquad\qquad t \;=\; \emptyset \\
\partial\;(\mathsf{tok}\;t') \qquad\qquad\quad t\;\textbf{with}\;t \equiv ?\;t' \\
\partial\;(\mathsf{tok}\;.t) \qquad\qquad\quad t \;\mid\; \mathsf{yes\;refl} \quad = \;\varepsilon \\
\partial\;(\mathsf{tok}\;t') \qquad\qquad\quad t \;\mid\; \mathsf{no}\;t \not\equiv t' \;=\; \emptyset \\
\partial\;(g_1 \mid g_2) \qquad\qquad\; t \;=\; \partial\;g_1\;t \mid \partial\;g_2\;t \\
\partial\;(\_\cdot\_\;\{\mathsf{true}\}\;\;g_1\;g_2)\;t \;=\; \\
\quad \partial\;g_1\;t \cdot \sharp?\;(\mathit{not}\;(\partial n\;g_1\;t))\;g_2 \mid \partial\;g_2\;t \\
\partial\;(\_\cdot\_\;\{\mathsf{false}\}\;g_1\;g_2)\;t \;=\; \\
\quad \partial\;g_1\;t \cdot \sharp?\;(\mathit{not}\;(\partial n\;g_1\;t))\;(\flat\;g_2)
\end{array}
$$

Almost
done

More examples

- Peter Hancock's examples from yesterday.
- Process calculi:
  Can avoid explicit support for
  (guarded) recursive definitions.

Conclusions

- Mixed induction/coinduction is fun.
- I encourage you to add this technique to your toolbox.

?