Subtyping, Declaratively

An Exercise in Mixed Induction and Coinduction

Nils Anders Danielsson Thorsten Altenkirch
(University of Nottingham)

Lac-Beauport, Québec, 2010-06-23

Introduction

- New way to define subtyping for recursive types.
- Example of the utility of mixed induction and coinduction ($\nu X.\mu Y.F\ X\ Y$).

Induction in Agda

Inductive types

$$\begin{aligned}&\textbf{data}\ \mathbb{N}\ :\ \mathit{Set}\ \textbf{where}\\&\quad \mathsf{zero}\ :\ \mathbb{N}\\&\quad \mathsf{suc}\ \ \ :\ \mathbb{N} \to \mathbb{N}\end{aligned}$$

$$\mathbb{N} \approx \mu X.\, 1 + X$$

Structural recursion:

$$\begin{aligned}&\_{+}\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}\\&\mathsf{zero} \quad + n = n\\&\mathsf{suc}\ m + n = \mathsf{suc}\ (m + n)\end{aligned}$$

Inductive types

Representation of (well-scoped) recursive types:

$$\begin{array}{lll}
\textbf{data}\ \mathit{Ty}\ (n : \mathbb{N}) : \mathit{Set}\ \textbf{where} & & \\
\quad \bot & : & \mathit{Ty}\ n \\
\quad \top & : & \mathit{Ty}\ n \\
\quad \mathsf{var} & : & \mathit{Fin}\ n \to \mathit{Ty}\ n \\
\quad _\multimap_ & : & \mathit{Ty}\ n \to \mathit{Ty}\ n \to \mathit{Ty}\ n \\
\quad \mu_\multimap_ & : & \mathit{Ty}\ (1+n) \to \mathit{Ty}\ (1+n) \to \mathit{Ty}\ n
\end{array}$$

$$\sigma, \tau ::= \bot \mid \top \mid X \mid \sigma \multimap \tau \mid \mu X.\, \sigma \multimap \tau$$

Inductive types

Representation of (well-scoped) recursive types:

- $\mu X.\, X \rightarrow X$:

  $$\begin{aligned} \sigma &: \mathit{Ty}\ 0 \\ \sigma &= \mu\ \mathsf{var}\ 0 \rightarrow \mathsf{var}\ 0 \end{aligned}$$

- $\mu X.\, (X \rightarrow \bot) \rightarrow \top$:

  $$\begin{aligned} \tau &: \mathit{Ty}\ 0 \\ \tau &= \mu\ (\mathsf{var}\ 0 \rightarrow \bot) \rightarrow \top \end{aligned}$$

Inductive types

Representation of (well-scoped) recursive types:

- Capture-avoiding substitution:

  $$\_[\_] \; : \; \mathit{Ty}\ (1 + n) \to \mathit{Ty}\ n \to \mathit{Ty}\ n$$

  $\sigma\ [\ \tau\ ]$: Replaces variable 0 in σ with τ.

Coinduction in Agda

Coinductive types

$$
\begin{array}{lll}
\textbf{data}\ \mathit{Tree} : \mathit{Set}\ \textbf{where} & & \\
\quad \bot & : & \mathit{Tree} \\
\quad \top & : & \mathit{Tree} \\
\quad \_\rhd\_ & : & \infty\ \mathit{Tree} \to \infty\ \mathit{Tree} \to \mathit{Tree}
\end{array}
$$

- ∞ marks coinductive arguments.
- $\mathit{Tree} \approx \nu X.\, 1 + 1 + X \times X$.
- Delay and force:

$$
\begin{array}{lll}
\sharp & : & A \to \infty\ A \\
\flat & : & \infty\ A \to A
\end{array}
$$

Coinductive types

Guarded corecursion:

$$
\begin{aligned}
&[\![\_]\!] \;:\; \textit{Ty 0} \rightarrow \textit{Tree} \\
&[\![\; \bot \;]\!] &&= \bot \\
&[\![\; \top \;]\!] &&= \top \\
&[\![\; \mathsf{var}\;()\;]\!] \\
&[\![\quad \sigma \rightarrowtriangle \tau \;]\!] &&= \sharp\,[\![\; \sigma \;]\!] \rightarrowtriangle \sharp\,[\![\; \tau \;]\!] \\
&[\![\; \mu\, \sigma \rightarrowtriangle \tau \;]\!] &&= [\![\; (\sigma \rightarrowtriangle \tau)\,[\, \mu\, \sigma \rightarrowtriangle \tau \,]\;]\!]
\end{aligned}
$$

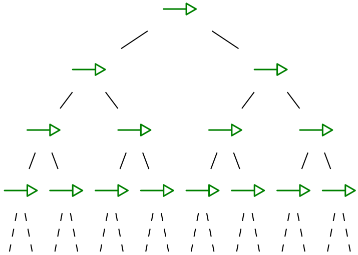
Coinductive types

Guarded corecursion:

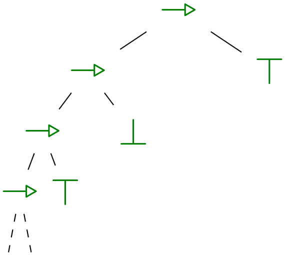
$$
\begin{array}{lcl}
[\![\_]\!] \;:\; \mathit{Ty}\ \mathit{0} \to \mathit{Tree} & & \\
[\![\ \bot\ ]\!] & = & \bot \\
[\![\ \top\ ]\!] & = & \top \\
[\![\ \mathsf{var}\ ()\ ]\!] & & \\
[\![\ \ \sigma \multimap \tau\ ]\!] & = & \sharp\, [\![\ \sigma\ ]\!] \multimap \sharp\, [\![\ \tau\ ]\!] \\
[\![\ \mu\, \sigma \multimap \tau\ ]\!] & = & \sharp\, [\![\ \sigma\ [\ \mu\, \sigma \multimap \tau\ ]\ ]\!] \multimap \\
 & & \sharp\, [\![\ \tau\ [\ \mu\, \sigma \multimap \tau\ ]\ ]\!]
\end{array}
$$

Coinductive types

$$[\![\ \mu\ \mathsf{var}\ 0 \rightarrow \mathsf{var}\ 0\ ]\!] =$$



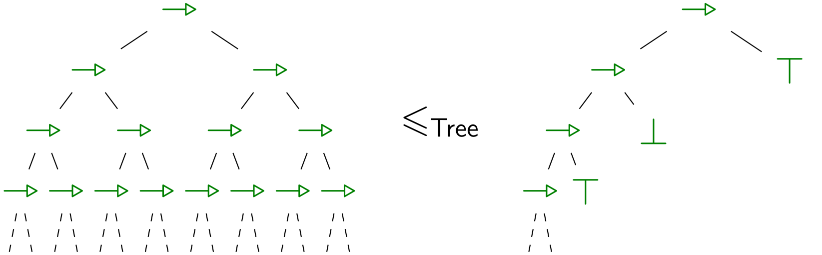
$$[\![\ \mu\ (\mathsf{var}\ 0 \rightarrow \bot) \rightarrow \top\ ]\!] =$$



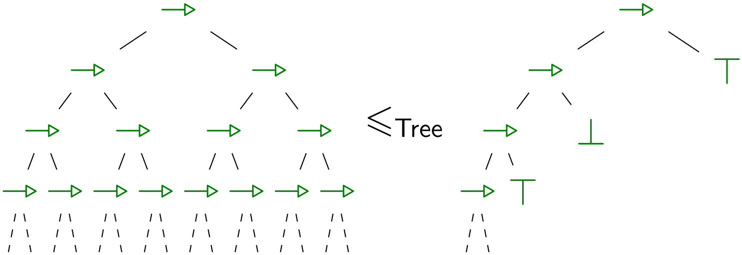
Subtyping

Subtyping

$$\mu\ \mathsf{var}\ 0 \rightarrow \mathsf{var}\ 0 \qquad \leqslant_{\mathsf{Type}} \quad \mu\ (\mathsf{var}\ 0 \rightarrow \bot) \rightarrow \top$$



Subtyping



$$\frac{}{\bot \leqslant_{\mathsf{Tree}} \tau} \qquad \frac{}{\sigma \leqslant_{\mathsf{Tree}} \top}$$

$$\frac{\flat\,\tau_1 \leqslant_{\mathsf{Tree}} \flat\,\sigma_1 \qquad \flat\,\sigma_2 \leqslant_{\mathsf{Tree}} \flat\,\tau_2}{\sigma_1 \to \sigma_2 \leqslant_{\mathsf{Tree}} \tau_1 \to \tau_2} \; \text{(coinductive)}$$

Indexed coinductive types

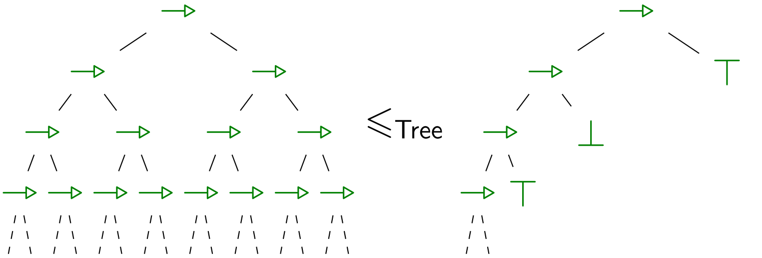
Inference system ≈ indexed data type:

$$
\begin{array}{lll}
\textbf{data}\ \_\leqslant_{\mathsf{Tree}}\_ & : & \mathit{Tree} \to \mathit{Tree} \to \mathit{Set}\ \textbf{where} \\
\quad \bot & : & \bot \leqslant_{\mathsf{Tree}} \tau \\
\quad \top & : & \sigma \leqslant_{\mathsf{Tree}} \top \\
\quad \_\rightarrow\_ & : & \infty\,(\flat\,\tau_1 \leqslant_{\mathsf{Tree}} \flat\,\sigma_1) \to \\
 & & \infty\,(\flat\,\sigma_2 \leqslant_{\mathsf{Tree}} \flat\,\tau_2) \to \\
 & & \sigma_1 \rightarrow \sigma_2 \leqslant_{\mathsf{Tree}} \tau_1 \rightarrow \tau_2
\end{array}
$$

Subtyping

$$\_\leqslant_{\mathsf{Type}}\_ \ :\ \mathit{Ty}\ 0 \to \mathit{Ty}\ 0 \to \mathit{Set}$$
$$\sigma \leqslant_{\mathsf{Type}} \tau \ =\ [\![\ \sigma\ ]\!] \leqslant_{\mathsf{Tree}} [\![\ \tau\ ]\!]$$

$$ex\ :\ \mu\ \mathsf{var}\ 0 \rightarrowtriangle \mathsf{var}\ 0 \leqslant_{\mathsf{Type}} \mu\ (\mathsf{var}\ 0 \rightarrowtriangle \bot) \rightarrowtriangle \top$$
$$ex\ =\ \sharp\ (\sharp\ ex \rightarrowtriangle \sharp\ \bot) \rightarrowtriangle \sharp\ \top$$



Subtyping

$$\_\leqslant_{\mathsf{Type}}\_ \;:\; \mathit{Ty}\ 0 \rightarrow \mathit{Ty}\ 0 \rightarrow \mathit{Set}$$
$$\sigma \leqslant_{\mathsf{Type}} \tau \;=\; [\![\, \sigma \,]\!] \leqslant_{\mathsf{Tree}} [\![\, \tau \,]\!]$$

Can we define this relation directly, without unfolding the types?

Declarative vs. algorithmic

Algorithmic Syntax-directed.

Declarative Explicit rules for high-level concepts: reflexivity, transitivity…

Declarative vs. algorithmic

Algorithmic Syntax-directed.

Declarative Explicit rules for high-level concepts: reflexivity, transitivity. . .

Algorithmic Less modular.

Declarative Problematic if coinductive.

Coinductive transitivity

Coinductive inference system with transitivity: trivial.

data _⩽_ : *Ty 0* → *Ty 0* → *Set* **where**
  . . .
  trans : ∞ (τ₁ ⩽ τ₂) → ∞ (τ₂ ⩽ τ₃) → τ₁ ⩽ τ₃

$$\dfrac{\dfrac{\dfrac{\vdots}{\sigma \leqslant \tau} \quad \dfrac{\vdots}{\tau \leqslant \tau}}{\sigma \leqslant \tau} \qquad \dfrac{\dfrac{\vdots}{\tau \leqslant \tau} \quad \dfrac{\vdots}{\tau \leqslant \tau}}{\tau \leqslant \tau}}{\sigma \leqslant \tau}$$

Stuck?

- Stuck with syntax-directed definition?
- No, can use mixed induction and coinduction.

Transitivity:	inductive
Remaining rules:	coinductive

Mixed induction and coinduction

data $\_\leqslant\_$: *Ty 0* → *Ty 0* → *Set* **where**

$\bot$: $\bot \leqslant \tau$

$\top$: $\sigma \leqslant \top$

$\_\rightharpoonup\_$: $\infty\,(\tau_1 \leqslant \sigma_1) \to \infty\,(\sigma_2 \leqslant \tau_2) \to$
$\sigma_1 \rightharpoonup \sigma_2 \leqslant \tau_1 \rightharpoonup \tau_2$

unfold : $\mu\,\tau_1 \rightharpoonup \tau_2 \leqslant (\tau_1 \rightharpoonup \tau_2)\,[\,\mu\,\tau_1 \rightharpoonup \tau_2\,]$

fold : $(\tau_1 \rightharpoonup \tau_2)\,[\,\mu\,\tau_1 \rightharpoonup \tau_2\,] \leqslant \mu\,\tau_1 \rightharpoonup \tau_2$

refl : $\tau \leqslant \tau$

trans : $\tau_1 \leqslant \tau_2 \to \tau_2 \leqslant \tau_3 \to \tau_1 \leqslant \tau_3$

Mixed induction and coinduction

$$
\begin{array}{l}
\textbf{data}\ \_\leqslant\_ \ :\ \mathit{Ty}\ 0 \to \mathit{Ty}\ 0 \to \mathit{Set}\ \textbf{where} \\
\quad \_\rightharpoonup\_ \ :\ \infty\,(\tau_1 \leqslant \sigma_1) \to \infty\,(\sigma_2 \leqslant \tau_2) \to \\
\qquad\qquad \sigma_1 \rightharpoonup \sigma_2 \leqslant \tau_1 \rightharpoonup \tau_2 \\
\quad \mathsf{trans}\ :\ \tau_1 \leqslant \tau_2 \to \tau_2 \leqslant \tau_3 \to \tau_1 \leqslant \tau_3
\end{array}
$$

$$
\begin{array}{l}
\_\leqslant\_ \ \approx\ \nu C.\ \mu I.\ \lambda\,\sigma\,\tau. \\
\qquad (\exists\,\sigma_1, \sigma_2, \tau_1, \tau_2.\ \sigma \equiv \sigma_1 \rightharpoonup \sigma_2 \times \tau \equiv \tau_1 \rightharpoonup \tau_2 \times \\
\qquad\qquad\qquad C\ \tau_1\ \sigma_1 \times C\ \sigma_2\ \tau_2) \\
\quad + (\exists\,\chi.\ I\ \sigma\ \chi \times I\ \chi\ \tau)
\end{array}
$$

Mixed induction and coinduction

$$
\begin{array}{lll}
\textbf{data}\ \_\leqslant\_ & : & \mathit{Ty}\ 0 \to \mathit{Ty}\ 0 \to \mathit{Set}\ \textbf{where} \\
\quad \bot & : & \bot \leqslant \tau \\
\quad \top & : & \sigma \leqslant \top \\
\quad \_\rightarrow\_ & : & \infty\,(\tau_1 \leqslant \sigma_1) \to \infty\,(\sigma_2 \leqslant \tau_2) \to \\
 & & \sigma_1 \rightarrow \sigma_2 \leqslant \tau_1 \rightarrow \tau_2 \\
\quad \mathsf{unfold} & : & \mu\,\tau_1 \rightarrow \tau_2 \leqslant (\tau_1 \rightarrow \tau_2)\,[\,\mu\,\tau_1 \rightarrow \tau_2\,] \\
\quad \mathsf{fold} & : & (\tau_1 \rightarrow \tau_2)\,[\,\mu\,\tau_1 \rightarrow \tau_2\,] \leqslant \mu\,\tau_1 \rightarrow \tau_2 \\
\quad \mathsf{refl} & : & \tau \leqslant \tau \\
\quad \mathsf{trans} & : & \tau_1 \leqslant \tau_2 \to \tau_2 \leqslant \tau_3 \to \tau_1 \leqslant \tau_3
\end{array}
$$

Equivalent to $\_\leqslant_{\mathsf{Type}}\_$.

Beware!

Partiality monad

$A_\bot$ Partial computations which may return something of type A.

$$
\begin{aligned}
&\textbf{data}\ \_{}_\bot\ (A : Set) : Set\ \textbf{where} \\
&\quad \mathsf{now} \ : A \to A_\bot \\
&\quad \mathsf{later} : \infty\,(A_\bot) \to A_\bot
\end{aligned}
$$

$$
\begin{aligned}
&never : A_\bot \\
&never = \mathsf{later}\ (\sharp\ never)
\end{aligned}
$$

Equality

When are two partial computations equivalent?

Strong bisimilarity (coinductive):

$$
\begin{array}{llr}
\textbf{data}\ \_\!\sim\!\_ \ : \ A_{\bot} \to A_{\bot} \to \mathit{Set}\ \textbf{where} & & \\
\quad \text{now} \ : & & \text{now}\ v \ \sim \ \text{now}\ v \\
\quad \text{later} \ : & \infty\,(\flat\, x \ \sim \ \flat\, y) \to & \text{later}\ x \ \sim \ \text{later}\ y
\end{array}
$$

Equality

When are two partial computations equivalent?

Weak bisimilarity (mixed):

$$
\begin{array}{llcl}
\textbf{data}\ \_\approx\_ & : A_{\bot} \to A_{\bot} \to \mathit{Set}\ \textbf{where} & & \\
\quad \mathsf{now} & : & & \mathsf{now}\ v \approx \mathsf{now}\ v \\
\quad \mathsf{later} & : \infty\,(\flat\, x \approx \flat\, y) & \to & \mathsf{later}\ x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{r}} & : x \approx \flat\, y & \to & x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{l}} & : \flat\, x \approx y & \to & \mathsf{later}\ x \approx y
\end{array}
$$

The problem of "weak bisimulation up to"

Weak bisimilarity is transitive. What happens if we make the definition more declarative?

$$
\begin{array}{llcl}
\multicolumn{4}{l}{\textbf{data}\ \_\approx\_\ :\ A_{\perp} \rightarrow A_{\perp} \rightarrow \mathit{Set}\ \textbf{where}} \\
\text{now} & : & & \text{now}\ v \approx \text{now}\ v \\
\text{later} & : & \infty\,(\flat\, x \approx \flat\, y) \rightarrow & \text{later}\ x \approx \text{later}\ y \\
\text{later}^{\mathrm{r}} & : & x \approx \flat\, y \rightarrow & x \approx \text{later}\ y \\
\text{later}^{\mathrm{l}} & : & \flat\, x \approx y \rightarrow & \text{later}\ x \approx y \\
\text{trans} & : & x \approx y \rightarrow y \approx z \rightarrow & x \approx z
\end{array}
$$

The problem of “weak bisimulation up to”

Weak bisimilarity is transitive. What happens if we make the definition more declarative?

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathit{trivial} : (x\ y : A_{\bot}) \to x \approx y} \\
\multicolumn{2}{l}{\mathit{trivial}\ x\ y =} \\
\quad x & \approx\langle\ \mathrm{later}^{\mathrm{r}}\ (\mathit{refl}\ x)\ \rangle \\
\quad \mathrm{later}\ (\sharp\ x) & \approx\langle\ \mathrm{later}\ (\sharp\ (\mathit{trivial}\ x\ y))\ \rangle \\
\quad \mathrm{later}\ (\sharp\ y) & \approx\langle\ \mathrm{later}^{\mathrm{l}}\ (\mathit{refl}\ y)\ \rangle \\
\quad y & \square
\end{array}
$$

The problem of "weak bisimulation up to"

Weak bisimilarity is transitive. What happens if we make the definition more declarative?

- Inductive case:
  Sound to postulate admissible rule.
- Coinductive case:
  Not always sound, proof may not be contractive.
- Known problem: "weak bisimulation up to".
- Subtyping unproblematic:
  $\_ \leqslant \_$ equivalent to $\_ \leqslant_{\mathsf{Type}} \_$.

Conclusions

- Mixed induction and coinduction is a useful technique.
- Declarative, mostly coinductive inference systems possible.
- In particular: subtyping for recursive types.
- But don't rely on intuitions which are only valid in the inductive case.

?