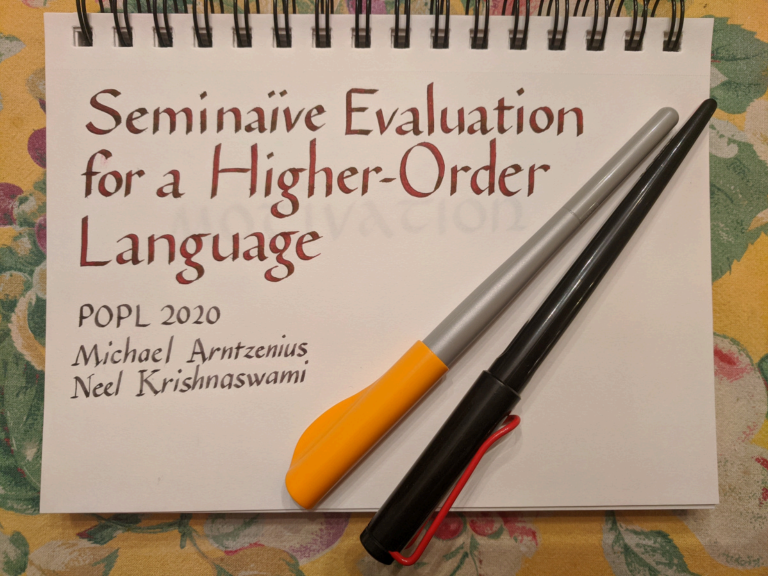

Seminaïve Evaluation
for a Higher-Order
Language
POPL 2020
Michael Arntzenius
Neel Krishnaswami

POP QUIZ

How can you compute each of the following?

- Graph reachability
- Regular expression matching
- Abstract interpretation

POP QUIZ

How can you compute each of the following?

- Graph reachability
- Regular expression matching
- Abstract interpretation

ANSWER

Iterate a monotone map to its fixed point.

$$reach(\mathrm{start}).$$

$$reach(Y) \leftarrow reach(X) \wedge edge(X, Y).$$

$$reach(\text{start}).$$
$$reach(Y) \leftarrow reach(X) \wedge edge(X, Y).$$
$$reach2(\text{start}).$$
$$reach2(Y) \leftarrow reach2(X) \wedge edge2(X, Y).$$
$$reach3(\text{start}).$$
$$reach3(Y) \leftarrow reach3(X) \wedge \textit{isnt-this-tedious}(X, Y).$$
$$reach4(\text{start}).$$
$$reach4(Y) \leftarrow reach4(X) \wedge \textit{yes-it-is-rather}(X, Y).$$

$$\mathit{reach} \;:\; \mathsf{Set}\,(\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set}\,\mathsf{Node}$$

$$\mathit{reach}\ \mathit{edge}\ \mathit{start} = \mathbf{fix}\ (\lambda R.\ \{\mathit{start}\} \cup \{y \mid x \in R, (x, y) \in \mathit{edge}\})$$

$$\textit{reach} : \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set}\ \mathsf{Node}$$
$$\textit{reach edge start} = \mathbf{fix}\ (\lambda R.\ \{\textit{start}\} \cup \{y \mid x \in R, (x, y) \in \textit{edge}\})$$

- a simply-typed λ-calculus with

$$reach : \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set}\ \mathsf{Node}$$

$$reach\ edge\ start = \mathbf{fix}\ (\lambda R.\ \{start\} \cup \{y \mid x \in R, (x, y) \in edge\})$$

- a simply-typed λ-calculus with
- a finite set datatype & set comprehensions,

$$reach : \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set}\ \mathsf{Node}$$

$$reach\ edge\ start = \mathbf{fix}\ (\lambda R.\ \{start\} \cup \{y \mid x \in R, (x, y) \in edge\})$$

- a simply-typed λ-calculus with
- a finite set datatype & set comprehensions,
- a monotone fixed point operator,

$$\textit{reach} : \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set}\ \mathsf{Node}$$

$$\textit{reach edge start} = \mathbf{fix}\ (\lambda R.\ \{\textit{start}\} \cup \{y \mid x \in R, (x, y) \in \textit{edge}\})$$

Datafun[ICFP 2016] is:

- a simply-typed λ-calculus with
- a finite set datatype & set comprehensions,
- a monotone fixed point operator,
- where *types are posets* and *all functions are monotone*,

$$reach : \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Node} \to \mathsf{Set\ Node}$$

$$reach\ edge\ start = \mathbf{fix}\ (\lambda R.\ \{start\} \cup \{y \mid x \in R, (x, y) \in edge\})$$

Datafun[ICFP 2016] is:

- a simply-typed λ-calculus with
- a finite set datatype & set comprehensions,
- a monotone fixed point operator,
- where *types are posets* and *all functions are monotone*,
- and non-monotonicity is handled via a comonad type $\Box A$.

The type $\Box A$ is defined:

$$x \in \Box A \iff x \in A$$
$$x \leqslant y : \Box A \iff x = y$$

Thus $f : \Box A \to B$ is monotone iff:

$$x = y \implies f(x) \leqslant f(y)$$

i.e., **always!**

MATH

$$
\begin{aligned}
\mathit{reach} &: \quad \text{Set}\,(\text{Node} \times \text{Node}) \;\to\; \text{Node} \to \text{Set Node} \\
\mathit{reach}\;\; \mathit{edge}\;\; \mathit{start} &= \\
&\quad \mathbf{fix}\;(\lambda R.\,\{\mathit{start}\} \cup \{y \mid x \in R, (x\,, y) \in \mathit{edge} \qquad \})
\end{aligned}
$$

DATAFUN

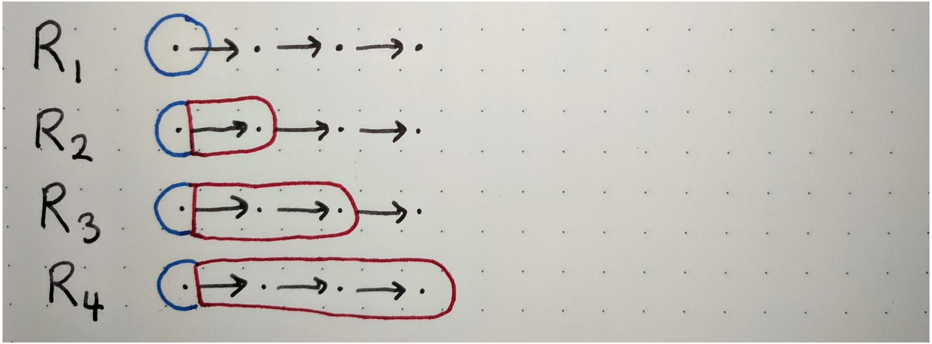
$$
\begin{aligned}
&\mathit{reach} : \Box(\mathsf{Set}\,(\mathsf{Node} \times \mathsf{Node})) \to \Box\mathsf{Node} \to \mathsf{Set}\,\mathsf{Node} \\
&\mathit{reach}\,[\mathit{edge}]\,[\mathit{start}] = \\
&\qquad \mathbf{fix}\,[\lambda R.\, \{\mathit{start}\} \cup \{y \mid x \in R, (x', y) \in \mathit{edge}, x = x'\}]
\end{aligned}
$$

$$\mathit{step}\ R = \{\mathit{start}\} \cup \{y \mid x \in R, (x, y) \in \mathit{edge}\}$$

$$\textit{step}\ R = \{\textit{start}\} \cup \{y \mid x \in R, (x, y) \in \textit{edge}\}$$

$$R_0 = \emptyset$$

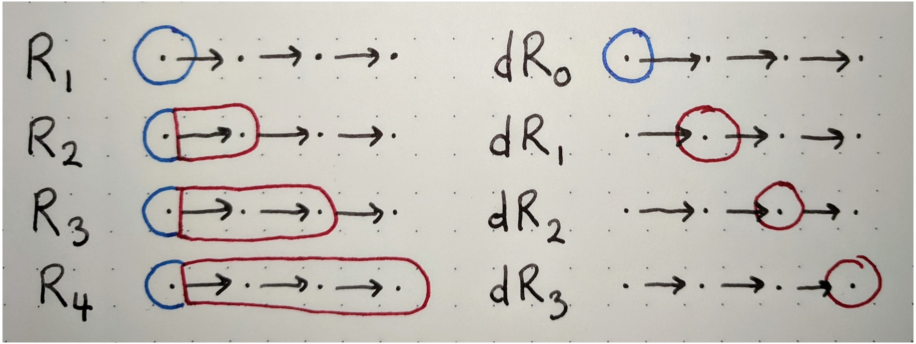
$$R_{i+1} = \textit{step}\ R_i$$



$$\textit{step}\ R = \{\textit{start}\} \cup \{y \mid x \in R, (x, y) \in \textit{edge}\}$$

$$R_0 = \emptyset \qquad dR_0 =$$

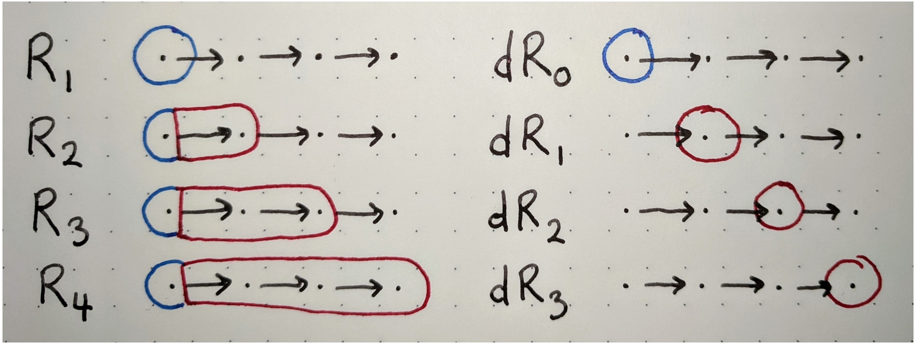
$$R_{i+1} = \textit{step}\ R_i \qquad dR_{i+1} =$$



$$step\ R = \{start\} \cup \{y \mid x \in R, (x, y) \in edge\}$$

$$R_0 = \emptyset \qquad dR_0 = \{start\}$$

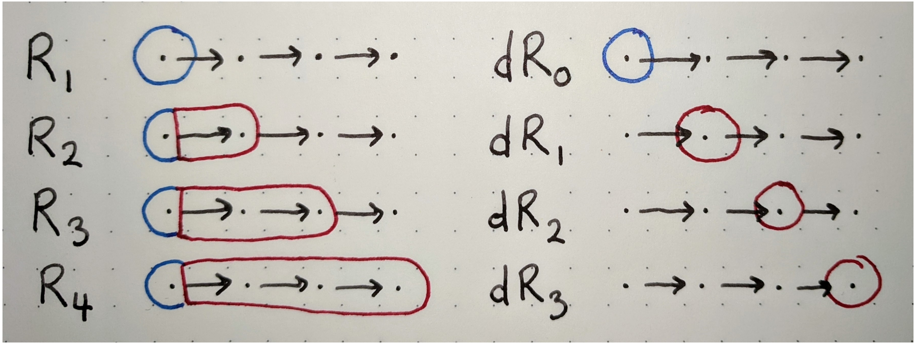
$$R_{i+1} = step\ R_i \qquad dR_{i+1} =$$



$$\mathit{step}\ R = \{\mathit{start}\} \cup \{y \mid x \in R, (x, y) \in \mathit{edge}\}$$

$$R_0 = \emptyset \qquad dR_0 = \{\mathit{start}\}$$

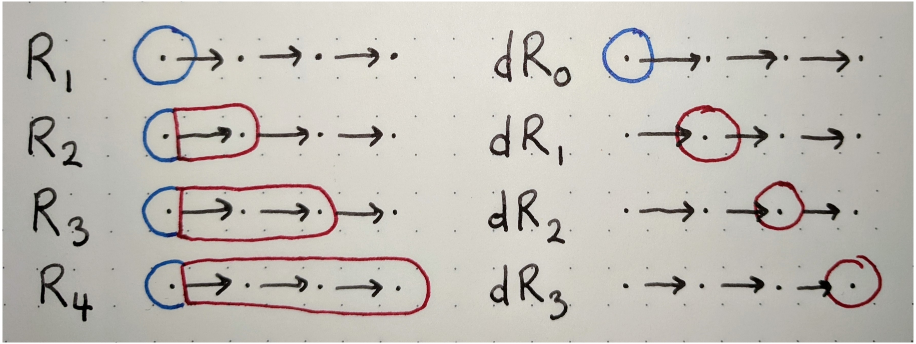
$$R_{i+1} = \mathit{step}\ R_i \qquad dR_{i+1} = \{y \mid x \in dR_i, (x, y) \in \mathit{edge}\}$$



$$\mathit{step}\ R = \{\mathit{start}\} \cup \{y \mid x \in R, (x, y) \in \mathit{edge}\}$$

$$R_0 = \emptyset \qquad dR_0 = \{\mathit{start}\}$$

$$R_{i+1} = R_i \cup dR_i \qquad dR_{i+1} = \{y \mid x \in dR_i, (x, y) \in \mathit{edge}\}$$



SEMINAÏVE EVALUATION

means

computing the changes between iterations

Incremental λ-calculus [Cai et al PLDI 2014, Giarrusso's thesis 2017, Giarrusso et al ESOP 2019]

Every type A has a *change type* ΔA and a relation $dx ::_A x \rightsquigarrow y$, where $x, y : A$ and $dx : \Delta A$.

$$\Delta(\mathsf{Set}\ A) = \mathsf{Set}\ A \qquad\qquad dx ::_{\mathsf{Set}\ A} x \rightsquigarrow x \cup dx$$

Incremental λ-calculus [Cai et al PLDI 2014, Giarrusso's thesis 2017, Giarrusso et al ESOP 2019]

Every type A has a *change type* ΔA and a relation $dx ::_A x \rightsquigarrow y$, where $x, y : A$ and $dx : \Delta A$.

$$\Delta(\text{Set } A) = \text{Set } A \qquad dx ::_{\text{Set } A} x \rightsquigarrow x \cup dx$$

$$\Delta(A \times B) = \Delta A \times \Delta B \qquad \frac{da ::_A a \rightsquigarrow a' \qquad db ::_B b \rightsquigarrow b'}{(da, db) ::_{A \times B} (a, b) \rightsquigarrow (a', b')}$$

Incremental λ-calculus [Cai et al PLDI 2014, Giarrusso's thesis 2017, Giarrusso et al ESOP 2019]

Every type A has a *change type* ΔA and a relation $dx ::_A x \rightsquigarrow y$, where $x, y : A$ and $dx : \Delta A$.

$$\Delta(\text{Set } A) = \text{Set } A \qquad dx ::_{\text{Set } A} x \rightsquigarrow x \cup dx$$

$$\Delta(A \times B) = \Delta A \times \Delta B \qquad \frac{da ::_A a \rightsquigarrow a' \qquad db ::_B b \rightsquigarrow b'}{(da, db) ::_{A \times B} (a, b) \rightsquigarrow (a', b')}$$

$$\Delta(\Box A) = 1 \qquad () ::_{\Box A} x \rightsquigarrow x$$

$$\Delta(A \to B) = \Box A \to \Delta A \to \Delta B$$

original input	change to input	change in output

$$\Delta(A \to B) = \Box A \to \Delta A \to \Delta B$$

original input — change to input — change in output

$$df \mathrel{::}_{A \to B} f \rightsquigarrow g \iff$$

$$\Delta(A \to B) = \Box A \to \Delta A \to \Delta B$$

original input — change to input — change in output

$$df \mathrel{::}_{A \to B} f \rightsquigarrow g \iff \frac{dx \mathrel{::}_A x \rightsquigarrow y}{\mathrel{::}_B f\,x \rightsquigarrow g\,y}$$

$$\Delta(A \to B) = \Box A \to \Delta A \to \Delta B$$

original input | change to input | change in output

$$df ::_{A \to B} f \rightsquigarrow g \iff \frac{dx ::_A x \rightsquigarrow y}{df\ x\ dx ::_B f\ x \rightsquigarrow g\ y}$$

$$\Delta(A \to B) = \Box A \to \Delta A \to \Delta B$$

original input — change to input — change in output

$$\mathrm{df} ::_{A \to B} f \rightsquigarrow f \iff \frac{\mathrm{dx} ::_A x \rightsquigarrow y}{\mathrm{df}\ x\ \mathrm{dx} ::_B f\ x \rightsquigarrow f\ y}$$

In this case, we call df the **derivative** of f.

$$\mathit{step} : \text{Set}\ A \to \text{Set}\ A$$

$$\mathit{step}' : \Box(\text{Set}\ A) \to \text{Set}\ A \to \text{Set}\ A$$

known world frontier new frontier

STRATEGY

$$R_i = \mathit{step}^i\ \emptyset$$

known world

$$\textit{step} : \text{Set } A \to \text{Set } A$$

$$\textit{step}' : \Box(\text{Set } A) \to \text{Set } A \to \text{Set } A$$

known world frontier new frontier

STRATEGY

$$R_i = \textit{step}^i\ \emptyset$$

known world

$$dR_i :: R_i \rightsquigarrow \textit{step}\ R_i$$

frontier

$$\textit{step} : \text{Set } A \to \text{Set } A$$

$$\textit{step}' : \Box(\text{Set } A) \to \text{Set } A \to \text{Set } A$$

known world frontier new frontier

STRATEGY

$$R_i = \textit{step}^i\ \emptyset \qquad dR_i :: R_i \rightsquigarrow \textit{step}\ R_i$$

known world frontier

IMPLEMENTATION

$$R_0 = \emptyset \qquad dR_0 =$$

$$R_{i+1} = \qquad dR_{i+1} =$$

$$\textit{step} : \text{Set } A \to \text{Set } A$$

$$\textit{step}' : \underbrace{\Box(\text{Set } A)}_{\text{known world}} \to \underbrace{\text{Set } A}_{\text{frontier}} \to \underbrace{\text{Set } A}_{\text{new frontier}}$$

STRATEGY

$$\underbrace{R_i}_{\text{known world}} = \textit{step}^i\ \emptyset \qquad \underbrace{dR_i}_{\text{frontier}} :: R_i \rightsquigarrow \textit{step}\ R_i$$

IMPLEMENTATION

$$R_0 = \emptyset \qquad dR_0 = \textit{step}\ \emptyset$$

$$R_{i+1} = \qquad dR_{i+1} =$$

$$step : \text{Set } A \to \text{Set } A$$

$$step' : \Box(\text{Set } A) \to \text{Set } A \to \text{Set } A$$

known world — frontier — new frontier

STRATEGY

$$R_i = step^i\ \emptyset \qquad dR_i :: R_i \rightsquigarrow step\ R_i$$

known world — frontier

IMPLEMENTATION

$$R_0 = \emptyset \qquad dR_0 = step\ \emptyset$$

$$R_{i+1} = R_i \cup dR_i \qquad dR_{i+1} =$$

$$step \ : \ \text{Set}\ A \to \text{Set}\ A$$

$$step' \ : \ \square(\text{Set}\ A) \to \text{Set}\ A \to \text{Set}\ A$$

known world frontier new frontier

STRATEGY

$$R_i = step^i\ \emptyset$$

known world

$$dR_i :: R_i \rightsquigarrow step\ R_i$$

frontier

IMPLEMENTATION

$$R_0 = \emptyset$$

$$R_{i+1} = R_i \cup dR_i$$

$$dR_0 = step\ \emptyset$$

$$dR_{i+1} = step'\ R_i\ dR_i$$

Two static transformations on expressions e:

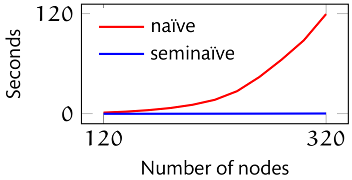
- ϕe annotates functions used by a fixed point with their derivatives.

  COMPLICATION: *We propagate derivatives by hijacking the $\Box$ comonad.*

- δe computes how ϕe changes as its free variables change.

  COMPLICATION: *Computing how set comprehensions change.*

- **Regex combinators!**
  as a use-case for higher-order functions.
- **Logical relations!**
  to prove ϕ/δ correct.
- **Further optimizations!**
  Must propagate $\emptyset$ to get asymptotic speedups.
- **Benchmarks!**



Ideas to take away:

- Bottom-up monotone fixed points are cool.
- To compute them efficiently, incrementalize the step function.
- Types let us control what things we need to incrementalize!

*un*FIN*ished*