Finding fixed points faster

Michael Arntzenius

University of Birmingham

HOPE @ ICFP 2018

Datalog + semi-naïve evaluation	$\subseteq$	Datafun + incremental λ-calculus

[1] Datalog
+
[2] semi-naïve evaluation

$\subseteq$

[3] Datafun
+
[4] incremental λ-calculus

Datalog

decidable logic programming

predicates = finite sets

Transitive closure of *edge*:

$$path(x, z) \leftarrow edge(x, z)$$
$$path(x, z) \leftarrow edge(x, y) \land path(y, z)$$

Transitive closure of *edge*, naïvely:

$$path_{i+1}(x, z) \leftarrow edge(x, z)$$
$$path_{i+1}(x, z) \leftarrow edge(x, y) \wedge path_{i}(y, z)$$

$$\vdots$$

$$
\begin{array}{c|l}
i = 3 & path_3(2,4) \leftarrow edge(2,3) \wedge path_2(3,4) \\
i = 4 & path_4(2,4) \leftarrow edge(2,3) \wedge path_3(3,4) \\
i = 5 & path_5(2,4) \leftarrow edge(2,3) \wedge path_4(3,4)
\end{array}
$$

$$\vdots$$

Wastefully re-deducing old facts makes me :(

Transitive closure of *edge*, seminaïvely:

$$\Delta path_0(x, z) \leftarrow edge(x, z)$$
$$\Delta path_{i+1}(x, z) \leftarrow edge(x, y) \wedge \Delta path_i(y, z)$$

$$path_{i+1}(x, y) \leftarrow path_i(x, y) \vee \Delta path_i(x, y)$$

Computes the changes **between** naïve iterations!

II. DATAFUN

$$path(x, z) \leftarrow edge(x, z)$$
$$path(x, z) \leftarrow edge(x, y) \wedge path(y, z)$$

$$\text{path} = \text{edge} \cup \{(x, z) \mid (x, y) \in \text{edge}, (y, z) \in \text{path}\}$$

Datalog

$$
\begin{aligned}
path(x,z) &\leftarrow edge(x,z) \\
path(x,z) &\leftarrow edge(x,y) \wedge path(y,z)
\end{aligned}
$$

Datafun

$$
\mathrm{path} = \mathrm{edge} \cup \{(x,z) \mid (x,y) \in \mathrm{edge}, (y,z) \in \mathrm{path}\}
$$

Datafun

- Simply-typed λ-calculus
- finite sets & monadic set comprehensions
- monotone[†] iterative fixed points

For more, see *Datafun: A functional Datalog* [ICFP '16]!

[†]Come to my poster presentation on Monday to learn about types for monotonicity!

$$\text{path} = \text{edge} \cup \{(x, z) \mid (x, y) \in \text{edge}, (y, z) \in \text{path}\}$$

$$\text{step } S = \text{edge} \cup \{(x, z) \mid (x, y) \in \text{edge}, (y, z) \in S\}$$
$$\text{path} = \textbf{fix} \text{ step}$$

$$\text{step } S = \text{edge} \cup \{(x, z) \mid (x, y) \in \text{edge}, (y, z) \in S\}$$
$$\text{path} = \textbf{fix } \text{step}$$

How do we compute ($\textbf{fix}\ f$), naïvely?

$$x_0 = \emptyset \qquad\qquad x_{i+1} = f(x_i)$$

Iterate until $x_i = x_{i+1}$.

Incremental λ-Calculus

"A Theory of Changes for Higher-Order Languages", PLDI '14
Yufei Cai, Paulo Giarrusso, Tillman Rendel, Klaus Ostermann

$$f : A \to B$$

$$\delta f : A \to \Delta A \to \Delta B$$

$$
\begin{aligned}
\mathsf{f} &: \mathsf{Set}\ A \to \mathsf{Set}\ A \\
\delta\mathsf{f} &: \mathsf{Set}\ A \to \mathsf{Set}\ A \to \mathsf{Set}\ A
\end{aligned}
$$

$$f : \mathrm{Set}\ A \to \mathrm{Set}\ A$$
$$\delta f : \mathrm{Set}\ A \to \mathrm{Set}\ A \to \mathrm{Set}\ A$$

$$\begin{aligned} x_0 &= \emptyset & dx_0 &= f\ \emptyset \\ x_{i+1} &= x_i \cup dx_i & dx_{i+1} &= \delta f\ x_i\ dx_i \end{aligned}$$

Theorem: $x_i = f^i\ x$

III. DETAILS AND COMPLICATIONS

Pick your poison!

1. Precise vs. cheap derivatives
2. Monotonicity and ordering
3. Sum types are tricky
4. Sets of functions are inefficient
5. Derivatives suck if you don't optimise them

For every type A

- a *change type* ΔA
- a *zero* function $\mathbf{0} : A \to \Delta A$
- and an *update* function $\oplus : A \to \Delta A \to A$

For every term $x : A \vdash M : B$,

- a derivative $x : A, dx : \Delta\Gamma \vdash \delta M : \Delta A$
- such that $M \oplus \delta M = M[(x \oplus dx)/x]$

1. Precise vs cheap derivatives

$$\delta(M \cup N) = \delta M \cup \delta N$$

vs

$$\delta(M \cup N) = (\delta M \setminus N) \cup (\delta N \setminus M)$$

2. Monotonicity and ordering

$$A \to B \;\; \textbf{vs} \;\; A \xrightarrow{+} B$$

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

$$\Delta(A \xrightarrow{+} B) = A \to \Delta A \xrightarrow{+} \Delta B$$

$$(dx \leqslant dy : \Delta A \iff (\forall a)\ a \oplus da \leqslant a \oplus db : A)?$$

Increasing changes only? What about incrementalizing Datafun?

Why do discrete functions need derivatives if their arguments can't change?

3. Sum types are tricky

$$\begin{aligned}\Delta(A + B) &= \Delta A \times \Delta B? \\ &= \Delta A \cup \Delta B? \\ &= \Delta A + \Delta B\end{aligned}$$

$$\begin{aligned}&\delta(\textbf{case } M \textbf{ of } \mathrm{in}_1\ x \to N_1; \mathrm{in}_2\ y \to N_2) \\ =\ &\textbf{case } (M, \delta M) \textbf{ of} \\ &\quad (\mathrm{in}_1\ x, \mathrm{in}_1\ dx) \to \delta N_1 \\ &\quad (\mathrm{in}_2\ y, \mathrm{in}_2\ dy) \to \delta N_2 \\ &\quad (\mathrm{in}_1\ x, \mathrm{in}_2\ dy) \to ??? \\ &\quad (\mathrm{in}_2\ x, \mathrm{in}_1\ dy) \to ???\end{aligned}$$

4. Sets of functions are inefficient

$$
\begin{aligned}
&\delta\,(\textstyle\bigcup(x \in M)\ N) \\
&= (\textstyle\bigcup(x \in \delta M)\ N) \\
&\cup\ (\textstyle\bigcup(x \in M \cup \delta M)\ \textbf{let}\ \mathrm{d}x = \mathbf{0}\ x\ \textbf{in}\ \delta N)
\end{aligned}
$$

4. Sets of functions are inefficient

$$\delta\,(\bigcup(x \in M)\ N)$$
$$=\ (\bigcup(x \in \delta M)\ N)$$
$$\cup\ \ (\bigcup(x \in M \cup \delta M)\ \textbf{let}\ dx = \mathbf{0}\ x\ \textbf{in}\ \delta N)$$

What is $(\mathbf{0}\ f)$ for $f : A \to B$?

It's the derivative of f.

5. Derivatives suck if you don't optimise them

$$
\begin{aligned}
X \cap Y &= \{x \mid x \in X, x \in Y\} \\
&= \bigcup(x \in X) \bigcup(y \in Y) \textbf{ if } x = y \textbf{ then } \{x\} \textbf{ else } \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\delta\,(\bigcup(x \in M)\ N) &= (\bigcup(x \in \delta M)\ N) \\
&\cup\ (\bigcup(x \in M \cup \delta M) \textbf{ let } dx = \mathbf{0}\ x \textbf{ in } \delta N)
\end{aligned}
$$

$$
\delta(X \cap Y) = \textit{horrible!}
$$

FIN