Finding fixed points faster

Michael Arntzenius

University of Birmingham

S-REPLS 5, 2016

$$\begin{array}{c} \text{Datalog} \\ + \\ \text{semi-naïve} \\ \text{evaluation} \end{array} \subseteq \begin{array}{c} \text{Datafun} \\ + \\ \text{incremental} \\ \lambda\text{-calculus} \end{array}$$

$$\begin{array}{ccc} {}^{1}\ \text{Datalog} & & {}^{3}\ \text{Datafun} \\ + & \subseteq & + \\ {}^{2}\ \text{semi-naïve evaluation} & & {}^{4}\ \text{incremental } \lambda\text{-calculus} \end{array}$$

Finding fixed points faster by static incrementalization

Datalog

decidable logic programming

predicates $=$ finite sets

```
% Transitive closure of ‘edge’.
path(X,Y) :- edge(X,Y).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

Naïve implementation

$$
\begin{aligned}
\text{step} &: \text{Set}\ (\text{Node} \times \text{Node}) \to \text{Set}\ (\text{Node} \times \text{Node}) \\
\text{step}\ \mathit{path} &= \{(x, y) \mid (x, y) \in \mathit{edge}\} \\
&\quad \cup \{(x, z) \mid (x, y) \in \mathit{edge}, (y, z) \in \mathit{path}\}
\end{aligned}
$$

$$
\begin{aligned}
\text{fix} &: (\alpha \to \alpha) \to \alpha \to \alpha \\
\text{fix}\ f\ \mathit{current} &= \mathbf{let}\ \mathit{next} = f\ \mathit{current}\ \mathbf{in} \\
&\quad\ \mathbf{if}\ \mathit{next} = \mathit{current}\ \mathbf{then}\ \mathit{current} \\
&\quad\ \mathbf{else}\ \text{fix}\ f\ \mathit{next}
\end{aligned}
$$

$$
\begin{aligned}
\text{path} &: \text{Set}\ (\text{Node} \times \text{Node}) \\
\text{path} &= \text{fix step}\ \emptyset
\end{aligned}
$$

Naïve implementation

$$
\begin{aligned}
\mathsf{step} &: \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \to \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \\
\mathsf{step}\ \mathit{path} &= \{(x, y) \mid (x, y) \in \mathit{edge}\} \\
&\quad \cup \{(x, z) \mid (x, y) \in \mathit{edge}, (y, z) \in \mathit{path}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{fix} &: (\alpha \to \alpha) \to \alpha \to \alpha \\
\mathsf{fix}\ f\ \mathit{current} &= \mathbf{let}\ \mathit{next} = f\ \mathit{current}\ \mathbf{in} \\
&\quad\ \mathbf{if}\ \mathit{next} = \mathit{current}\ \mathbf{then}\ \mathit{current} \\
&\quad\ \mathbf{else}\ \mathsf{fix}\ f\ \mathit{next}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{path} &: \mathsf{Set}\ (\mathsf{Node} \times \mathsf{Node}) \\
\mathsf{path} &= \mathsf{fix}\ \mathsf{step}\ \emptyset
\end{aligned}
$$

Unnecessary recomputation!

Semi-naïve implementation

$$\begin{aligned} &\text{small-step} \;:\; \text{Set } (\text{Node} \times \text{Node}) \longrightarrow \text{Set } (\text{Node} \times \text{Node}) \\ &\text{small-step } \mathit{new} = \{(x,z) \mid (x,y) \in \mathit{edges}, (y,z) \in \mathit{new}\} \end{aligned}$$

Semi-naïve implementation

$$\text{small-step} \;:\; \text{Set}\;(\text{Node} \times \text{Node}) \rightarrow \text{Set}\;(\text{Node} \times \text{Node})$$
$$\text{small-step}\; \mathit{new} = \{(x, z) \mid (x, y) \in \mathit{edges}, (y, z) \in \mathit{new}\}$$

$$\text{fix-faster} : (\text{Set}\,\alpha \rightarrow \text{Set}\,\alpha \rightarrow \text{Set}\,\alpha) \rightarrow \text{Set}\,\alpha \rightarrow \text{Set}\,\alpha \rightarrow \text{Set}\,\alpha$$

$$\begin{aligned}
&\text{fix-faster}\; f\; \mathit{current}\; \mathit{new} = \\
&\quad \mathbf{let}\; \mathit{to\text{-}add} = f\; \mathit{current}\; \mathit{new}\; \mathbf{in} \\
&\quad \mathbf{if}\; \mathit{to\text{-}add} \subseteq \mathit{current}\; \mathbf{then}\; \mathit{current} \\
&\quad \mathbf{else}\; \text{fix-faster}\; f\; (\mathit{current} \cup \mathit{to\text{-}add})\; \mathit{to\text{-}add}
\end{aligned}$$

Semi-naïve implementation

$$\text{small-step} \;:\; \text{Set}\ (\text{Node} \times \text{Node}) \to \text{Set}\ (\text{Node} \times \text{Node})$$
$$\text{small-step}\ \mathit{new} = \{(x, z) \mid (x, y) \in \mathit{edges}, (y, z) \in \mathit{new}\}$$

$$\text{fix-faster} : (\text{Set}\ \alpha \to \text{Set}\ \alpha \to \text{Set}\ \alpha) \to \text{Set}\ \alpha \to \text{Set}\ \alpha \to \text{Set}\ \alpha$$
$$\text{fix-faster}\ f\ \mathit{current}\ \mathit{new} =$$
$$\quad \mathbf{let}\ \mathit{to\text{-}add} = f\ \mathit{current}\ \mathit{new}\ \mathbf{in}$$
$$\quad \mathbf{if}\ \mathit{to\text{-}add} \subseteq \mathit{current}\ \mathbf{then}\ \mathit{current}$$
$$\quad \mathbf{else}\ \text{fix-faster}\ f\ (\mathit{current} \cup \mathit{to\text{-}add})\ \mathit{to\text{-}add}$$

$$\text{path} \;:\; \text{Set}\ (\text{Node} \times \text{Node})$$
$$\text{path} = \text{fix-faster}\ (\lambda x\ dx.\ \text{small-step}\ dx)\ \mathit{edge}\ \mathit{edge}$$

$$\mathsf{fix} : (\alpha \to \alpha) \to \alpha \to \alpha$$

as iteration, not laziness

Datafun

- Simply-typed λ-calculus
- with iterative fixed points
- and monotonicity typing

(and other stuff, see ICFP'16 paper)

Datalog:

```
path(X,Y) :- edge(X,Y).
path(X,Z) :- edge(X,Y), path(Y,Z).
```

Datafun:

$$\textbf{fix } \mathit{path} \textbf{ is } \mathit{edge} \cup \{(x, z) \mid (x, y) \in \mathit{edge}, (y, z) \in \mathit{path}\}$$

Naïve implementation strategy for

$$\textbf{fix } \mathit{path} \textbf{ is } \mathit{edge} \cup \{(x,z) \mid (x,y) \in \mathit{edge}, (y,z) \in \mathit{path}\}$$

is

$$\text{fix } (\lambda\, \mathit{path}.\; \mathit{edge} \cup \{(x,z) \mid (x,y) \in \mathit{edge}, (y,z) \in \mathit{path}\})\; \emptyset$$

using our iterative 'fix' function from earlier.

Is there an analogue of faster-fix**?**

Incremental λ-Calculus

"A Theory of Changes for Higher-Order Languages", PLDI'14

Yufei Cai, Paulo Giarrusso, Tillman Rendel, Klaus Ostermann

For every type A

- a *change type* ΔA
- and operator $\oplus : A \to \Delta A \to A$.

Given term $f : A \to B$

- $\delta f : A \to \Delta A \to \Delta B$
- such that $f (x \oplus dx) = f\ x \oplus \delta f\ x\ dx$

if one knows $v = f\ x$, often cheaper to compute RHS!

$$(\lambda x\, dx.\ \text{small-step}\ dx) \approx \delta(\text{step})\ \textbf{!}$$

$(\lambda x\, dx.\ \text{small-step}\ dx) \approx \delta(\text{step})$!

$$\begin{aligned}\text{step}\ \mathit{path} = {} & \{(x, y) \mid (x, y) \in \mathit{edge}\} \\ & \cup \{(x, z) \mid (x, y) \in \mathit{edge}, (y, z) \in \mathit{path}\}\end{aligned}$$

$$\text{small-step} : \text{Set}\ (\text{Node} \times \text{Node}) \to \text{Set}\ (\text{Node} \times \text{Node})$$
$$\text{small-step}\ \mathit{new} = \{(x, z) \mid (x, y) \in \mathit{edges}, (y, z) \in \mathit{new}\}$$

$(\lambda x\, dx.\ \text{small-step}\ dx) \approx \delta(\text{step})$!

$$\begin{aligned}\text{step}\ path = & \{(x, y) \mid (x, y) \in edge\} \\ & \cup \{(x, z) \mid (x, y) \in edge, (y, z) \in path\}\end{aligned}$$

$$\begin{aligned}\text{small-step} &: \text{Set}\ (\text{Node} \times \text{Node}) \rightarrow \text{Set}\ (\text{Node} \times \text{Node}) \\ \text{small-step}\ new &= \{(x, z) \mid (x, y) \in edges, (y, z) \in new\}\end{aligned}$$

Find fixed points faster by static incrementalization!

$$
\begin{array}{l}
\text{faster-fix} \ : \ (\alpha \to \Delta\alpha \to \Delta\alpha) \to \alpha \to \Delta\alpha \to \alpha \\
\text{faster-fix}\ \mathit{df}\ \mathit{current}\ \mathit{change} = \\
\quad \textbf{let}\ \mathit{next} = \mathit{current} \oplus \mathit{change}\ \textbf{in} \\
\quad \textbf{if}\ \mathit{next} \leq \mathit{current}\ \textbf{then}\ \mathit{current} \\
\quad \textbf{else}\ \text{faster-fix}\ \mathit{df}\ \mathit{next}\ (\mathit{df}\ \mathit{current}\ \mathit{change})
\end{array}
$$

(I have a proof in my notes that this slide is too small to contain.)

Applying this to Datafun

- Monotonicity $\to$ increasing changes only!
  $\Delta(\text{Set } \alpha) = \text{Set } \alpha$
- $\Delta(\Box A) = 1$? No!
  Zero-changes are not trivial!
- $\delta(\bigvee(x \in e_1)\ e_2)$?
  In particular, if $e_1 : \text{Set } (A \to B)$.