Datafun

Michael Arntzenius[1] Neel Krishnaswami[2]

[1]University of Birmingham

[2]University of Cambridge

ICFP 2016

```
[redacted]
    print c        # can be replaced by print 0
    print x        # but this can't
    [redacted]
```

```
x := 0
[redacted]
  print c
  print x
  [redacted]
```

$x = 0$

```
x := 0
c := x
[redacted]
  print c
  print x
  [redacted]
```

$x = 0$

$x = 0, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = 0, c = 0$
`  print c`	
`  print x`	
[redacted]	

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = 0, c = 0$
`  print c`	$x = 0, c = 0$
`  print x`	
[redacted]	

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = 0, c = 0$
`  print c`	$x = 0, c = 0$
`  print x`	$x = 0, c = 0$
[redacted]	

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = 0, c = 0$
`  print c`	$x = 0, c = 0$
`  print x`	$x = 0, c = 0$
`  x += 1`	$x = 1, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = 0, c = 0$
`  print c`	$x = 0, c = 0$
`  print x`	$\mathbf{x = 0}, c = 0$
`  x += 1`	$x = 1, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = 0, c = 0$
`  print x`	$x = 0, c = 0$
`  x += 1`	$x = 1, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = \top, c = 0$
`  print x`	$x = 0, c = 0$
`  x += 1`	$x = 1, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = \top, c = 0$
`  print x`	$x = \top, c = 0$
`  x += 1`	$x = 1, c = 0$

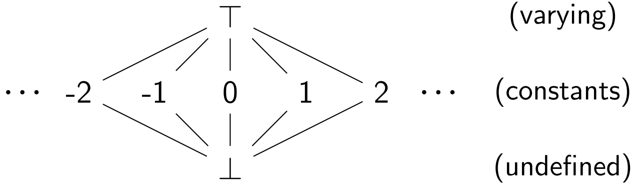
`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = \top, c = 0$
`  print x`	$x = \top, c = 0$
`  x += 1`	$x = \top, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = \top, c = 0$
`  print x`	$x = \top, c = 0$
`  x += 1`	$x = \top, c = 0$

`x := 0`	$x = 0$
`c := x`	$x = 0, c = 0$
`while true do`	$x = \top, c = 0$
`  print c`	$x = \top, c = 0$
`  print x`	$x = \top, c = 0$
`  x += 1`	$x = \top, c = 0$

Compute **fixed points**
of **monotone maps**
on **semilattices**
satisfying an **ascending chain condition**

- **Fixed point**: Keep going until nothing changes.
- **Monotone**: Unidirectional: $\bot \Rightarrow$ constant $\Rightarrow \top$
- **Semilattice**:



- **ACC**: Can't go up forever.

Examples of computing **fixed points** of **monotone maps** on **semilattices** satisfying an **ascending chain condition**:

- Static analyses
- Graph algorithms: reachability, shortest path, ...
- Parsing context-free grammars
- Datalog (as long as the semilattice is finite sets)

Datafun is:

- a simply typed λ-calculus
- where **types are posets**
  & some are semilattices
- that **tracks monotonicity via types**
- to let you compute **fixed points**
- and know they terminate.

Types as posets

Type	**Meaning**	**Ordering**
$\mathbb{N}$	naturals	$0 < 1 < 2 < \dots$
2	booleans	false $<$ true
$\{A\}$	finite subsets of A	$\subseteq$
$A \to B$	functions	pointwise
$A \xrightarrow{+} B$	**monotone** functions	pointwise

Types as posets

Type	Meaning	Ordering
$\mathbb{N}$	naturals	$0 < 1 < 2 < \dots$
2	booleans	false $<$ true
$\{A\}$	finite subsets of A	$\subseteq$
$A \to B$	functions	pointwise
$A \xrightarrow{+} B$	**monotone** functions	pointwise

$$\mathit{member} \;:\; \mathbb{N} \to \{\mathbb{N}\} \xrightarrow{+} 2$$
$$\mathit{member}\; x\; \mathbf{s} = \exists (y \in \mathbf{s})\; x = y$$

Tracking monotonicity

- Two types of *function*: discrete or monotone
- Two kinds of *variable*: discrete or monotone
- Two *typing contexts*: Δ discrete, Γ monotone

$$\Delta; \Gamma \vdash e : A$$

"e has type A with free variables Δ, Γ; moreover, e is monotone in Γ."

Tracking monotonicity: Function application

$$\frac{\Delta; \Gamma \vdash f : A \xrightarrow{+} B \qquad \Delta; \Gamma \vdash a : A}{\Delta; \Gamma \vdash f\ a : B} \ \textsc{monotone app}$$

Tracking monotonicity: Function application

$$\frac{\Delta; \Gamma \vdash f : A \xrightarrow{+} B \qquad {\color{red}\Delta; \Gamma} \vdash a : A}{\Delta; \Gamma \vdash f\ a : B} \textsc{ monotone app}$$

$$\frac{\Delta; \Gamma \vdash f : A \rightarrow B \qquad {\color{red}\Delta; \emptyset} \vdash a : A}{\Delta; \Gamma \vdash f\ a : B} \textsc{ discrete app}$$

Tracking monotonicity: Function application

$$\frac{\Delta; \Gamma \vdash f : A \xrightarrow{+} B \qquad \color{red}{\Delta; \Gamma} \vdash a : A}{\Delta; \Gamma \vdash f\ a : B} \text{ MONOTONE APP}$$

$$\frac{\Delta; \Gamma \vdash f : A \rightarrow B \qquad \color{red}{\Delta; \emptyset} \vdash a : A}{\Delta; \Gamma \vdash f\ a : B} \text{ DISCRETE APP}$$

Otherwise:

$$\begin{aligned} \mathit{coerce} &: (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \xrightarrow{+} \mathbb{N}) \\ \mathit{coerce}\ f\ \mathbf{x} &= f\ \color{red}{\mathbf{x}} \end{aligned}$$

Tracking monotonicity: Finite sets

$$e \ ::= \ \ldots \mid \{\} \mid e \cup e \mid \{e\} \mid \bigcup(x \in e)\, e$$

Tracking monotonicity: Finite sets

$$e ::= \ldots \mid \{\} \mid e \cup e \mid \{e\} \mid \bigcup(x \in e)\, e$$

$$\frac{\Delta; \emptyset \vdash e : A}{\Delta; \Gamma \vdash \{e\} : \{A\}}$$

Tracking monotonicity: Finite sets

$$e ::= \ldots \mid \{\} \mid e \cup e \mid \{e\} \mid \bigcup(x \in e)\, e$$
$$\mid \quad \{e \mid x \in e, \ldots\}$$

$$\frac{\Delta; \emptyset \vdash e : A}{\Delta; \Gamma \vdash \{e\} : \{A\}}$$

Example: Relational composition

$$(\bullet) : \{A \times \underset{eq}{B}\} \xrightarrow{+} \{\underset{eq}{B} \times C\} \xrightarrow{+} \{A \times C\}$$

$$\mathbf{s} \bullet \mathbf{t} = \{(x, z) \mid (x, y) \in \mathbf{s}, (!y, z) \in \mathbf{t}\}$$

Fixed points

fix $\mathbf{x}$ is e

Fixed points

$$\text{fix } \mathbf{x} \text{ is } e$$

$$\frac{\Delta; \Gamma, \mathbf{x} : \underset{fin}{L} \vdash e : \underset{fin}{L}}{\Delta; \Gamma \vdash \text{fix } \mathbf{x} \text{ is } e : \underset{fin}{L}}$$

A **monotone map** on a **finite semilattice with decidable equality**.

Fixed points

$$\mathsf{fix}\ \mathbf{x}\ \mathsf{is}\ e$$

$$\frac{\Delta; \Gamma, \mathbf{x} : \underset{fin}{L} \vdash e : \underset{fin}{L}}{\Delta; \Gamma \vdash \mathsf{fix}\ \mathbf{x}\ \mathsf{is}\ e : \underset{fin}{L}}$$

A **monotone map** on a **finite semilattice with decidable equality**.

Example: Reachability

$$
\begin{aligned}
\mathit{path} &: \{\underset{\mathit{fin}}{A} \times \underset{\mathit{fin}}{A}\} \xrightarrow{+} \{\underset{\mathit{fin}}{A} \times \underset{\mathit{fin}}{A}\} \\
\mathit{path}\ \mathbf{E} &= \mathsf{fix}\ \mathbf{P}\ \mathsf{is}\ \mathbf{E} \cup (\mathbf{P} \bullet \mathbf{P})
\end{aligned}
$$

In Datalog:

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

Example: CYK Parsing

Nonterminals: $A, B, C\ldots$
Literal strings: $s, t, \ldots$

Rules are all of the form $A \longrightarrow B\ C$ or $A \longrightarrow s$.

Example: CYK Parsing

Nonterminals: $A, B, C \ldots$
Literal strings: $s, t, \ldots$

Rules are all of the form $A \rightarrow B\ C$ or $A \rightarrow s$.

Apply the following inference rules **to saturation**:

Example: CYK Parsing

Nonterminals: $A, B, C \ldots$
Literal strings: $s, t, \ldots$

Rules are all of the form $A \longrightarrow B\ C$ or $A \longrightarrow s$.

Apply the following inference rules **to saturation**:

$$\frac{A \longrightarrow s \quad w[i..j] = s}{A(i,j)} \qquad \frac{A \longrightarrow B\ C \quad B(i,j) \quad C(j,k)}{A(i,k)}$$

where $A(i,j) =$ "A produces the substring $w[i..j]$"
and w is the input string

Example: CYK Parsing

$$\textbf{type } \text{rule} = \textsc{Concat}(\text{symbol}, \text{symbol}) \mid \textsc{String}(\text{string})$$
$$\textbf{type } \text{grammar} = \{\text{symbol} \times \text{rule}\}$$
$$\textbf{type } \text{fact} = \text{symbol} \times \mathbb{N} \times \mathbb{N}$$

$$\text{step} \;:\; \text{string} \to \text{grammar} \to \{\text{fact}\} \xrightarrow{+} \{\text{fact}\}$$

$$\begin{aligned}
&\text{step } w\ G\ \mathbf{prev} = \\
&\qquad \{(a, i, k) \mid (a, \textsc{Concat}(b, c)) \in G, \\
&\qquad\qquad\qquad (!b, i, j) \in \mathbf{prev}, (!c, !j, k) \in \mathbf{prev}\} \\
&\quad \cup \{\, (a, i, i + \text{length } s) \\
&\qquad \mid (a, \textsc{String}(s)) \in G, \\
&\qquad\ \ i \in \text{range } 0\ (\text{length } w - \text{length } s), \\
&\qquad\ \ s = \text{substring } w\ i\ (i + \text{length } s)\}
\end{aligned}$$

Summary

- Many algorithms are concisely expressed as **fixed points** of **monotone maps** on **semilattices**.
- Datafun is a simple, pure, and total language for computing these fixed points.
- Key idea: **track monotonicity with types!**
- Has a simple denotational semantics (in paper) & prototype implementation (on github).
- Generalizes Datalog to other semilattices.

`rntz.net/datafun`

Fin.

Future work

- Optimization
  - Semi-naïve evaluation
  - Dataflow (push & pull)
  - Magic sets
- More semilattice types
- More flexible termination/ACC checking
- Aggregation operations (summing, averaging)
  - Commutative monoids?
- Other applications of types for monotonicity
  - Types for functoriality?
  - LVars & monotone processes

Datafun vs Datalog

Datafun pros	**Datalog pros**
Functional abstraction!	Finiteness/ACC is automatic
Semilattices other than $\mathcal{P}_{\text{fin}}$	Often more concise
Can do arithmetic	Existing optimization literature
Can nest sets	

See also: FLIX, PLDI 2016, Madsen et al

Datafun vs Flix

Datafun pros	Flix **pros**
Functions on relations	Programmer-defined semilattices
Types for monotonicity	No types for monotonicity

Boolean elimination in a monotone world

$$\frac{\Delta; \emptyset \vdash e_1 : 2 \qquad \Delta; \Gamma \vdash e_2 : A \qquad \Delta; \Gamma \vdash e_3 : A}{\Delta; \Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : A}\ \textsc{Discrete If}$$

$$\frac{\Delta; \Gamma \vdash e_1 : 2 \qquad \Delta; \Gamma \vdash e_2 : L}{\Delta; \Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ \varepsilon : L}\ \textsc{Monotone If}$$

For example:

$$\begin{aligned} \mathsf{guard} &: 2 \xrightarrow{+} \{A\} \xrightarrow{+} \{A\} \\ \mathsf{guard}\ \mathbf{c}\ \mathbf{s} &= \mathsf{if}\ \mathbf{c}\ \mathsf{then}\ \mathbf{s}\ \mathsf{else}\ \{\} \end{aligned}$$