Let a Thousand Flowers Bloom

An Algebraic Representation of Edge Graphs

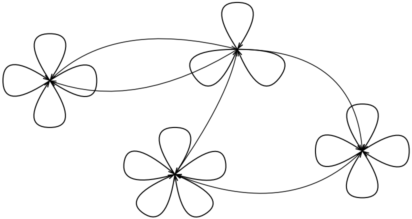
Jack Liell-Cock [1] Tom Schrijvers [2]

[1] University of Oxford

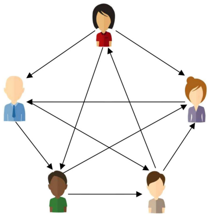
[2] KU Leuven

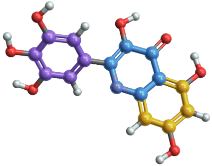
<Programming> 2024
13th March, 2024

Introduction

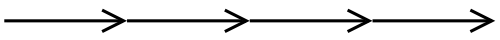


Beyond illustrating thousands of flowers blooming, graphs are ubiquitous for representing networks in computer systems

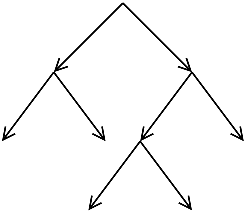




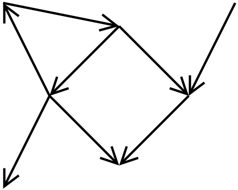
The Problem



A list



A tree

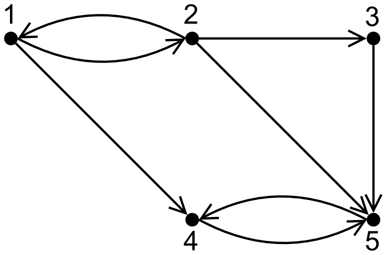


A graph

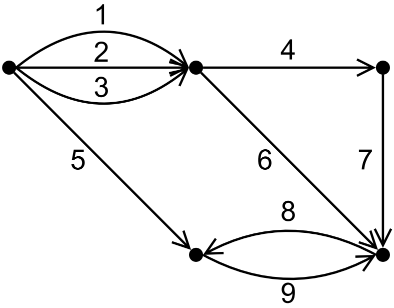
- Graphs evade capture as a *total algebraic* data type
- Common algebraic data types such as lists and trees have *directional* structure
- *Homogeneity* of graphs leave many frameworks riddled with partial functions, verbose interfaces, or complex run times

Background

Graphs



Node graph



Edge graph

Algebras

An algebra is an underlying set along with some constants and operators (i.e. 1 and $\times$) and equational laws that all terms satisfy (i.e. $a \times 1 = a$)

Example

A *monoid* $(X, \varepsilon, \oplus)$ is a set X with constant $\varepsilon \in X$ and binary operator $\oplus$ such that for all $x, y, z \in X$

- $x \oplus \varepsilon = x$
- $\varepsilon \oplus x = x$
- $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

The integers $\mathbb{Z}$ with $\oplus = +$ and $\varepsilon = 0$ forms a monoid.

Algebraic Datatypes

Construction primitives for injection, constants and composition

```
data List a =
        Singleton a
      | Empty
      | List a :++: List a
```

Equational laws for equivalent constructions

- `a :++: Empty = a`
- `Empty :++: a = a`
- `(a :++: b) :++: c = a :++: (b :++: c)`

So a list is a *monoid* with $\varepsilon = \texttt{Empty}$ and binary operator `:++:`

Algebraic Datatypes

- Computations become recursive functions (which respect the equational laws)
- For example, we can sum a list of integers because $(\mathbb{Z}, 0, +)$ satisfies the list axioms

```
sum :: List Int -> Int
sum (Singleton n) = n
sum Empty         = 0
sum (a :++: b)    = sum a + sum b
```

- The recursion scheme derives naturally from the construction primitives

Node Graphs
Constructors

Node graph algebra [Mokhov, 2017]

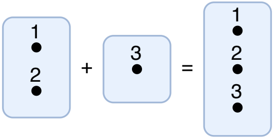
```
data NodeGraph a =
      Empty
    | Node a
    | Overlay (NodeGraph a) (NodeGraph a)
    | Connect (NodeGraph a) (NodeGraph a)
```

For graphs $R = (N, E)$ where N is the set of nodes and $E \subseteq N \times N$ are the edges:

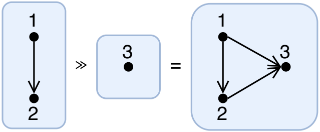
- Empty: $\varepsilon = (\emptyset, \emptyset)$
- Node: $\dot{x} = (\{x\}, \emptyset)$
- Overlay: $(N, E) + (N', E') = (N \cup N', E \cup E')$
- Connect: $(N, E) \gg (N', E') = (N \cup N', E \cup E' \cup N \times N')$

Node Graph

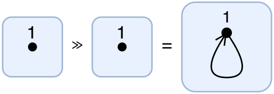
Examples



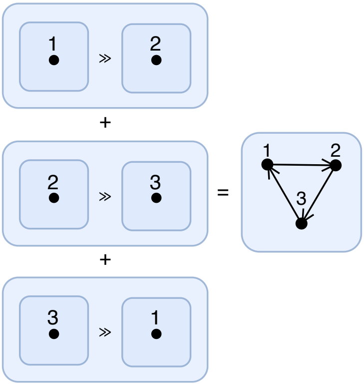
Overlay operator



Connect operator



Petal graph

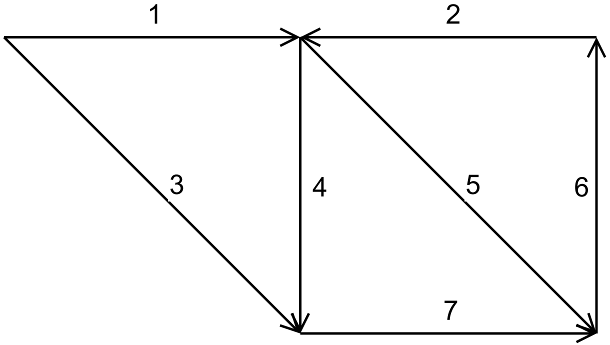


Loop graph

Our Work

Flow Representation

Example



$$\{(\emptyset, \{1,3\}),\ (\{1,2\}, \{4,5\}),\ (\{6\}, \{2\}),\ (\{3,4\}, \{7\}),\ (\{5,7\}, \{6\})\}$$

Definition

A *flow representation* for edges E is a subset $\gamma \subseteq \mathbb{P}(E) \times \mathbb{P}(E)$ such that

- $\bigcup_{x \in \gamma} \pi_1 x = E$ and $\bigcup_{x \in \gamma} \pi_2 x = E$
- $\forall x \neq y \in \gamma,\ \pi_1 x \cap \pi_1 y = \emptyset$ and $\pi_2 x \cap \pi_2 y = \emptyset$
- $(\emptyset, \emptyset) \notin \gamma$

where π_i are the projections and $\mathbb{P}$ is the powerset operator. The set of all flow representations is Γ.

Flow Representation
Equivalence

A multigraph representation G is a tuple (N, E, σ, τ) where

- N is the set of nodes
- E is the set of edges
- $\sigma : E \rightarrow N$ selects the source node for each edge
- $\tau : E \rightarrow N$ selects the target node for each edge

Theorem 1

The multigraph representation quotiented by the relation $\sim$, which identifies graphs up to node renaming and adding & removing of isolated nodes, is isomorphic to the flow representation:

$$G/\sim \;\cong\; \Gamma$$

Edge Graphs

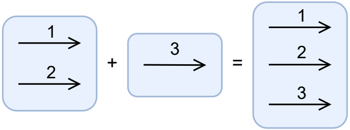
Constructors

A similar story is followed for the edge graph construction

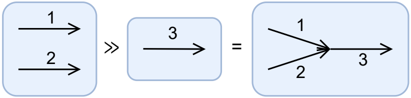
```
data EdgeGraph a =
      Empty
    | Edge a
    | Overlay (EdgeGraph a) (EdgeGraph a)
    | Into (EdgeGraph a) (EdgeGraph a)
    | Pits (EdgeGraph a) (EdgeGraph a)
    | Tips (EdgeGraph a) (EdgeGraph a)
```

Edge Graphs

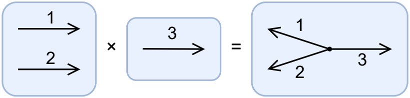
Examples



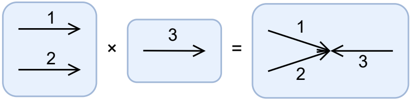
Overlay operator



Into operator



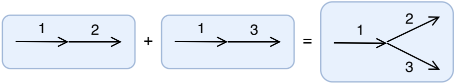
Pits operator



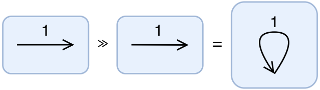
Tips operator

Edge Graphs

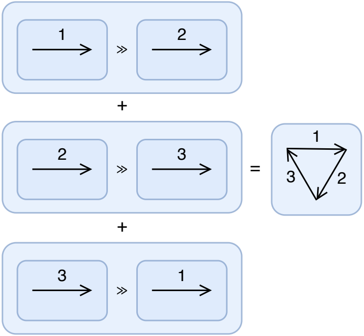
Examples



Overlay example



Petal graph



Loop graph

Theorem 2

There is a partial order on the flow representation with least upper bounds.

The edge graph constructors are defined in the flow representation as

- Empty: $\varepsilon = \emptyset$
- Edge: $\vec{x} = \{(\emptyset, \{x\}), (\{x\}, \emptyset)\}$
- Overlay: $a + b = \mathrm{lub}(a, b)$
- Into: $a \gg b = \mathrm{lub}(a, b, \dots)$
- Tips: $a \times b = \mathrm{lub}(a, b, \dots)$
- Pits: $a \diamond b = \mathrm{lub}(a, b, \dots)$

Edge Graphs

Axioms

- $(\Gamma, \varepsilon, +)$ is a commutative, idempotent monoid
- $(\Gamma, \varepsilon, \diamond)$ is a commutative monoid
- $(\Gamma, \varepsilon, \times)$ is a commutative monoid
- $(\Gamma, \varepsilon, \gg)$ is a monoid
- $\diamond, \times, \gg$ distribute over $+$
- 2 decomposition axiom schemas
- 2 reflexivity axioms
- 6 transitivity axioms

Theorem 3

The axioms are sound and complete with respect to the flow representation: two graph terms are equal when interpreted as flow representations if and only if they can be proved equal via the axioms.

Definition

A function $h : \Gamma \to A$ is an *edge graph homomorphism* if there is a constant $e \in A$, a function $f : E \to A$ and four binary operators $o, i, p, t : A \times A \to A$ such that

$$\begin{aligned}
h(\varepsilon) &= e \\
h(\vec{x}) &= f(x) \\
h(a + b) &= o(h(a), h(b)) \\
h(a \gg b) &= i(h(a), h(b)) \\
h(a \diamond b) &= p(h(a), h(b)) \\
h(a \times b) &= t(h(a), h(b))
\end{aligned}$$

and preserving the edge graph axioms.

Homomorphisms

Examples

- By soundness and completeness, all well-formed edge graph algorithms are edge graph homomorphisms
- Determining a graph algorithm amounts to finding a model that captures the solution properties
- Simple examples:

Homomorphism	ε	$\overset{\rightarrow}{\sqcup}$	$+$	$\gg$	$\diamond$	$\times$
Identity $\Gamma \to \Gamma$	ε	$\overset{\rightarrow}{\sqcup}$	$+$	$\gg$	$\diamond$	$\times$
Underlying Edges $\Gamma \to E$	$\emptyset$	$\{\sqcup\}$	$\cup$	$\cup$	$\cup$	$\cup$
Transpose $\Gamma \to \Gamma$	ε	$\overset{\rightarrow}{\sqcup}$	$+$	$\ll$	$\times$	$\diamond$

where $a \ll b = b \gg a$

Homomorphisms

Shortest Paths

```haskell
data End a = Pit a | Tip a deriving (Eq, Ord)
type ShortestPaths a = Map (End a, End a) a

h :: (Ord a, Num a) => EdgeGraph a -> ShortestPaths a
h Empty          = Map.empty
h (Edge x)       = Map.fromList [
        ((Pit x, Pit x), 0),
        ((Pit x, Tip x), x),
        ((Tip x, Tip x), 0)]
h (Overlay x y)  = closure (Map.unionWith min (h x) (h y))
h (Into x y)     = closure (connect Tip Pit (h x) (h y))
h (Tips x y)     = closure (connect Pit Pit (h x) (h y))
h (Pits x y)     = closure (connect Tip Tip (h x) (h y))
```

Summary

We have:

- Introduced a novel representation for edge graphs
- Categorised this representation as an algebra
- Implemented a graph data type using this algebraic interface
- Implemented common graph algorithms as homomorphisms

Main results:

- Node-agnostic multigraph rep. is isomorphic to flow rep.
- Flow rep. has partial order with least upper bounds
- Edge graph axioms are sound & complete for flow rep.

References I

Mokhov, A. (2017).

Algebraic graphs with class (functional pearl).

In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, Haskell 2017, pages 2–13, New York, NY, USA. Association for Computing Machinery.

Appendix A: Edge Graph Axioms

Appendix A: Edge Graph Axioms

- $(\Gamma, \varepsilon, +)$ is a commutative, idempotent monoid
- $(\Gamma, \varepsilon, \diamond)$ is a commutative monoid
- $(\Gamma, \varepsilon, \times)$ is a commutative monoid
- $(\Gamma, \varepsilon, \gg)$ is a monoid
- $\diamond, \times, \gg$ distribute over $+$
- Decomposition axioms: for $\square$ and $\blacksquare$ any of $\gg$, $\diamond$ and $\times$

$$
\begin{aligned}
a \square (b \blacksquare c) &= a \square b + a \square c + b \blacksquare c, \\
(a \square b) \blacksquare c &= a \square b + a \blacksquare c + b \blacksquare c.
\end{aligned}
$$

- Reflexivity axioms:

$$
\begin{aligned}
\vec{x} \diamond \vec{x} &= \vec{x}, \\
\vec{x} \times \vec{x} &= \vec{x}.
\end{aligned}
$$

Appendix A: Edge Graph Axioms

- Transitivity axioms: for all $a \neq \varepsilon$,

$$
\begin{aligned}
(a \diamond b) + (a \diamond c) &= a \diamond b \diamond c, \\
(b \gg a) + (a \diamond c) &= b \gg (a \diamond c), \\
(a \gg b) + (a \gg c) &= a \gg (b \diamond c), \\
(a \times b) + (a \gg c) &= (a \times b) \gg c, \\
(b \gg a) + (c \gg a) &= (b \times c) \gg a, \\
(a \times b) + (a \times c) &= a \times b \times c.
\end{aligned}
$$

Appendix B: Node Graph Axioms

Appendix B: Node Graph Axioms

- $(R, \varepsilon, +)$ is a commutative, idempotent monoid
  - $a + (b + c) = (a + b) + c$
  - $a + b = b + a$
  - $a + \varepsilon = a$
  - $a + a = a$
- $(R, \gg, \varepsilon)$ is a monoid
  - $a \gg (b \gg c) = (a \gg b) \gg c$
  - $\varepsilon \gg a = a$
  - $a \gg \varepsilon = a$
- $\gg$ distributes over $+$
  - $a \gg (b + c) = a \gg b + a \gg c$
  - $(a + b) \gg c = a \gg c + b \gg c$
- The decomposition axiom
  - $a \gg b \gg c = a \gg b + a \gg c + b \gg c$

Note identity and idempotency of $+$ can be derived from the remaining axioms

Appendix B: Node Graph Axioms

For refined graph classes, additional axioms can be introduced

- Reflexive graphs

$$\dot{x} \gg \dot{x} = \dot{x}$$

- Undirected graphs

$$a \gg b = b \gg a$$

- Transitive graphs

$$\forall b \neq \varepsilon, a \gg b + b \gg c = a \gg b \gg c$$

- Hypergraphs - replace decomposition axiom with

$$a \gg b \gg c \gg d =$$
$$a \gg b \gg c + a \gg b \gg d + a \gg c \gg d + b \gg c \gg d$$