CS 477: Dataflow Analysis and Abstract Interpretation

Sasa Misailovic

Based on previous slides by Saman Amarasinghe, Martin Rinard, and by Vikram Adve and Martin Vechev

University of Illinois at Urbana-Champaign

Partial Orders

Set P

Partial order relation $\leq$ such that $\forall x,y,z \in P$

- $x \leq x$ (reflexive)
- $x \leq y$ and $y \leq x$ implies $x = y$ (antisymmetric)
- $x \leq y$ and $y \leq z$ implies $x \leq z$ (transitive)

Can use partial order to define

- Upper and lower bounds
- Least upper bound
- Greatest lower bound

Upper Bounds

If $S \subseteq P$ then

- $x \in P$ is an upper bound of S if $\forall y \in S.\ y \leq x$
- $x \in P$ is the least upper bound of S if
  - x is an upper bound of S, and
  - $x \leq y$ for all upper bounds y of S
- **$\vee$ - join**, least upper bound, **lub**, supremum, **sup**
  - $\vee S$ is the least upper bound of S
  - $x \vee y$ is the least upper bound of $\{x,y\}$

Lower Bounds

If $S \subseteq P$ then

- $x \in P$ is a lower bound of S if $\forall y \in S.\ x \leq y$
- $x \in P$ is the greatest lower bound of S if
  - x is a lower bound of S, and
  - $y \leq x$ for all lower bounds y of S
- **$\wedge$ - meet**, greatest lower bound, **glb**, infimum, **inf**
  - $\wedge S$ is the greatest lower bound of S
  - $x \wedge y$ is the greatest lower bound of $\{x,y\}$

Covering

$x < y$ if $x \leq y$ and $x \neq y$

x is covered by y (y covers x) if

- $x < y$, and
- $x \leq z < y$ implies $x = z$

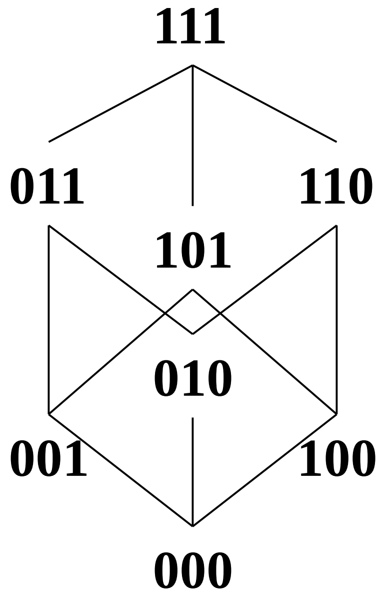
Conceptually, y covers x if there are no elements between x and y

Example

P = { 000, 001, 010, 011, 100, 101, 110, 111}

(standard boolean lattice, also called **hypercube**)

x ≤ y **is equivalent to** (x bitwise-and y) = x



Hasse Diagram

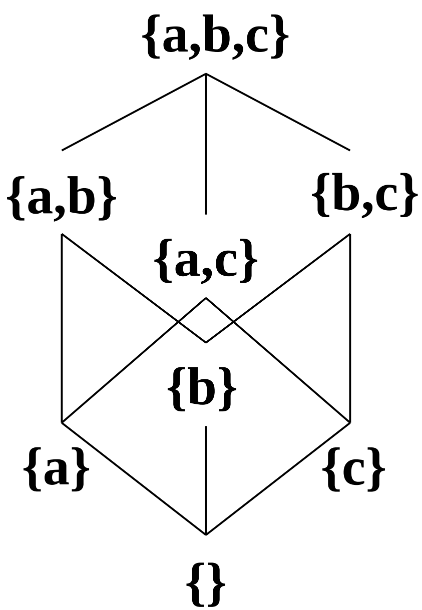
- If y covers x
  - Line from y to x
  - y above x in diagram

Example: Same as

$P = \wp(\{a, b, c\})$

(standard powerset lattice, also called **hypercube**)

$x \leq y$ **is equivalent to** $x \subseteq y = x$



Hasse Diagram

- If y covers x
  - Line from y to x
  - y above x in diagram

Lattices

Consider poset $(P, \leq)$ and the operators $\wedge$ (meet) and $\vee$ (join)

If for all $x,y \in P$ there exist $x \wedge y$ <u>and</u> $x \vee y$,
then P is a **lattice**.

If for all $S \subseteq P$ there exist $\wedge S$ and $\vee S$
then P is a <u>**complete**</u> **lattice**.

All **finite** lattices are **complete**

Example of a lattice that is not complete: Integers Z

- For any $x, y \in Z$, $x \vee y = \max(x,y)$, $x \wedge y = \min(x,y)$
- But $\vee Z$ and $\wedge Z$ do not exist
- $Z \cup \{+\infty, -\infty\}$ is a complete lattice

Top and Bottom

Greatest element of P (if it exists) is top ($\top$)

- $\forall a \in L . a \vee \top = \top$
- Note: $\forall a \in L .\ a \leq \top$ and $\top \wedge a = a$

Least element of P (if it exists) is bottom ($\perp$)

- $\forall a \in L .\ a \wedge \perp = \perp$
- Note: $\forall a \in L .\ \perp \leq a$ and $\perp \vee a = a$

Lattice (Recap)

$$(P, \leq, \wedge, \vee, \perp, \top)$$

- Set
- Partial order
- Meet
- Join
- Bottom
- Top

Connection Between $\leq$, $\wedge$, and $\vee$

Theorem: The following 3 properties are equivalent:

- $x \leq y$
- $x \vee y = y$
- $x \wedge y = x$

Let's prove:

- $x \leq y$ implies $x \vee y = y$ and $x \wedge y = x$
- $x \vee y = y$ implies $x \leq y$
- $x \wedge y = x$ implies $x \leq y$

Then by transitivity, we can obtain

- $x \vee y = y$ implies $x \wedge y = x$
- $x \wedge y = x$ implies $x \vee y = y$

Connecting Lemma Proofs

Lemma: $x \leq y$ implies $x \vee y = y$

Proof:

- $x \leq y$ implies y is an upper bound of $\{x,y\}$.
- Any upper bound z of $\{x,y\}$ must satisfy $y \leq z$.
- So y is least upper bound of $\{x,y\}$ and $x \vee y = y$

Lemma: $x \leq y$ implies $x \wedge y = x$

Proof:

- $x \leq y$ implies x is a lower bound of $\{x,y\}$.
- Any lower bound z of $\{x,y\}$ must satisfy $z \leq x$.
- So x is greatest lower bound of $\{x,y\}$ and $x \wedge y = x$

Connecting Lemma Proofs

Lemma: $x \vee y = y$ implies $x \leq y$

Proof:

- y is an upper bound of $\{x,y\}$ implies $x \leq y$

Lemma: $x \wedge y = x$ implies $x \leq y$

Proof:

- x is a lower bound of $\{x,y\}$ implies $x \leq y$

Lattices as Algebraic Structures

We have previously defined $\vee$ and $\wedge$ in terms of $\leq$

We will now define $\leq$ in terms of $\vee$ and $\wedge$

- Start with $\vee$ and $\wedge$ as arbitrary algebraic operations that satisfy ***associative, commutative, idempotence, and absorption*** laws
- We will define $\leq$ using $\vee$ and $\wedge$
- We will show that $\leq$ is a partial order

Intuitive concept of $\vee$ and $\wedge$ as information combination operators (or, and) or set operations (union, intersection)

Algebraic Properties of Lattices

Assume arbitrary operations $\vee$ and $\wedge$ such that

- $(x \vee y) \vee z = x \vee (y \vee z)$ (associativity of $\vee$)
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (associativity of $\wedge$)
- $x \vee y = y \vee x$ (commutativity of $\vee$)
- $x \wedge y = y \wedge x$ (commutativity of $\wedge$)
- $x \vee x = x$ (idempotence of $\vee$)
- $x \wedge x = x$ (idempotence of $\wedge$)
- $x \vee (x \wedge y) = x$ (absorption of $\vee$ over $\wedge$)
- $x \wedge (x \vee y) = x$ (absorption of $\wedge$ over $\vee$)

Connection Between $\wedge$ and $\vee$

Thm: $x \vee y = y$ if and only if $x \wedge y = x$

Proof ('if'): $x \vee y = y \;\; => \;\; x = x \wedge y$

$$\begin{aligned} x &= x \wedge (x \vee y) && \text{(by absorption)} \\ &= x \wedge y && \text{(by assumption)} \end{aligned}$$

Proof ('only if'): $x \wedge y = x \;\; => \;\; y = x \vee y$

$$\begin{aligned} y &= y \vee (y \wedge x) && \text{(by absorption)} \\ &= y \vee (x \wedge y) && \text{(by commutativity)} \\ &= y \vee x && \text{(by assumption)} \\ &= x \vee y && \text{(by commutativity)} \end{aligned}$$

Properties of $\leq$

Define: $x \leq y$ if $x \vee y = y$

Thm : $x \leq y$ is a partial order

Proof of transitive property. Must show that

$$x \vee y = y \text{ and } y \vee z = z \text{ implies } x \vee z = z$$

$x \vee z = x \vee (y \vee z)$ (by assumption)

$= (x \vee y) \vee z$ (by associativity)

$= y \vee z$ (by assumption)

$= z$ (by assumption)

Properties of $\leq$

Proof of asymmetry property. Must show that

$$x \vee y = y \text{ and } y \vee x = x \text{ implies } x = y$$

$x = y \vee x$ (by assumption)

$\quad = x \vee y$ (by commutativity)

$\quad = y$ (by assumption)

Proof of reflexivity property. Must show that

$$x \vee x = x\text{, which follows directly}$$

$x \vee x = x$ (by idempotence)

Properties of $\leq$

Induced operation $\leq$ agrees with original definitions of $\vee$ and $\wedge$, i.e.,

- $x \vee y = \sup \{x, y\}$
- $x \wedge y = \inf \{x, y\}$

Proof of $x \vee y = \sup \{x, y\}$

Consider any upper bound u for x and y.

Given $x \vee u = u$ and $y \vee u = u$, must show $x \vee y \leq u$, i.e., $(x \vee y) \vee u = u$

$$\begin{aligned} u &= x \vee u && \text{(by assumption)} \\ &= x \vee (y \vee u) && \text{(by assumption)} \\ &= (x \vee y) \vee u && \text{(by associativity)} \end{aligned}$$

Proof of $x \wedge y = \inf \{x, y\}$

- Consider any lower bound L for x and y.
- Given $x \wedge L = L$ and $y \wedge L = L$, must show $L \leq x \wedge y$, i.e., $(x \wedge y) \wedge L = L$

$L = x \wedge L$ (by assumption)

$= x \wedge (y \wedge L)$ (by assumption)

$= (x \wedge y) \wedge L$ (by associativity)

Semi-lattice $(P, \wedge)$

Set P and binary operation $\wedge$ such that $\forall x,y,z \in P$

- $x \wedge x = x$ (idempotent)
- $x \wedge y = y \wedge x$ implies $x = y$ (commutative)
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (associative)

The operation $\wedge$ imposes a partial order on P

If $((L, \leq), \wedge, \vee)$ is a lattice, then

- $(L, \wedge)$ is a **meet semi-lattice**
- $(L, \vee)$ is a **join semi-lattice**

Give us more flexibility to define the analysis.

- Since our analyses deal with complete lattices, we will represent the framework on them, but it can also be defined on semi-lattices
- Some dataflow analyses can be only represented on semi-lattices

Chains

A **poset (S, ≤) is a chain** if $\forall x,y \in S \,.\, y \leq x$ or $x \leq y$

Height of a poset/lattice: the size of the maximum chain.

(S, ≤) is finite if it has the finite height.

P satisfies the ***ascending chain condition*** if for all sequences $x_1 \leq x_2 \leq \ldots$ there exists n such that $x_n = x_{n+1} = \ldots$

- When a particular ascending chain has the property that $x_n = x_{n+1} = \ldots$ we say that it stabilizes
- Then ascending chain condition means that all ascending chains stabilize

From one variable to more

If L is a poset then so is the Cartesian product L x L:

Let $(L_1, \leq_1)$ and $(L_2, \leq_2)$ be posets.

Then $(L^*, \leq^*)$ is also a poset, where

$\top$

$L^* = \{\ (l_1, l_2)\ \mid\ l_1 \in L_1, l_2 \in L_2\}$ and $(l_{11}, l_{21}) \leq^* (l_{12}, l_{22})$

iff $l_{11} \leq_1 l_{12}$ and $l_{21} \leq_2 l_{22}$

This construction extends immediately on lattices, so that for $S \subseteq L^*$, we define $\perp^* = (\perp_1, \perp_2)$, we define

$$glb(Y) = (glb\ \{\ l_1\ |(l_1, \_\ ) \in \mathrm{Y}\ \},\ glb\ \{\ l_2\ |(\ \_\ , l_2) \in Y\}$$

and same for lub and $\top^*$

See Nielsen, Nielsen and Hankin book

From one variable to more

Total function space (S -> L) :

Let $(L, \leq)$ be a poset, S a set and f total function. Then $(L^f, \leq^f)$ is also a poset, where

$L^f = \{f: S \to L\}$ and $f' \leq^f f''$ iff $\forall s \in S \,.\, f'(s) \leq f''(s)$.

To extend to lattices, we define $\perp^f = \lambda s \,.\, \perp$ and $glb(Y) = \lambda s \,.\, glb_0 \{ f(s) \mid f \in Y \}$ and same for lub and $\top^f$

Monotone Function Space (L_1 -> L_2) :

Let $(L_1, \leq_1)$ and $(L_2, \leq_2)$ be posets and f monotone. Then $(L^f, \leq^f)$ is also a poset, where $\perp^f = \lambda s \,.\, \perp_2$ and

$L^f = \{f: L_1 \to L_2\}$ and $f' \leq^f f''$ iff $\forall l_1 \in L_1 \,.\, f'(l_1) \leq_2 f''(l_1)$

Application to Dataflow Analysis

Dataflow information will be lattice values

- **Transfer functions** operate on lattice values
- Solution algorithm will generate **increasing sequence of values** at each program point
- Ascending chain condition will ensure **termination**

We will use $\vee$ to combine values at control-flow join points

Transfer Functions

Transfer function **f:** $P \rightarrow P$ is defined for each node in control flow graph

- Maps lattice elements to lattice elements

The function **f** models effect of the node on the program information

Transfer Functions

Each dataflow analysis problem has a **set F of transfer functions** $f: P \rightarrow P$. This set F contains:

- **Identity function** belongs to the set, $i \in F$
- F must be **closed under composition**:
  $\forall f,g \in F.$ the function $h = \lambda x.f(g(x)) \in F$
- Each $f \in F$ must be **monotonic**:
  $x \leq y$ implies $f(x) \leq f(y)$
- Sometimes all $f \in F$ are **distributive***:
  $f(x \vee y) = f(x) \vee f(y)$
- Note that Distributivity implies monotonicity

*One can also define distributivity in terms of $\wedge$ ("meet"): $f(x \wedge y) = f(x) \wedge f(y)$

Distributivity Implies Monotonicity

Proof.*

Assume distributivity: $f(x \vee y) = f(x) \vee f(y)$

Must show: $x \vee y = y$ implies $f(x) \vee f(y) = f(y)$

$$\begin{aligned} f(y) &= f(x \vee y) && \text{(by assumption)} \\ &= f(x) \vee f(y) && \text{(by distributivity)} \end{aligned}$$

*For $f(x \wedge y) = f(x) \wedge f(y)$, show $x \wedge y = x \Rightarrow f(x) \wedge f(y) = f(x)$; $f(x) = f(x \wedge y) = f(x) \wedge f(y)$

Knaster-Tarsky Fixed-point Theorem

Let:

- $(L, \leq, \wedge, \vee, \top, \bot)$ be a complete lattice
- $f : L \rightarrow L$ be a monotonic function
- ***fix (f)*** is the set of fixed points of f

The set ***fix (f)*** with relation $\leq$, and operators $\wedge$, $\vee$ is forming a complete lattice.

- There will be a least fixed-point and greatest fixed point

Consequences:

- f has at least one fixpoint
- That fixpoint is the largest element in the chain

$$\bot, f(\bot), f(f(\bot)), f(f(f(\bot))), \ldots , f^n(\bot)$$

Putting the Pieces Together…

Forward Dataflow Analysis

Simulates execution of program forward with flow of control

Tuple **(G, (L, ≤), F, I)** – (graph, (lattice), transfer fs., initial val.)

For each node $n \in$ **G**, we have

- in_n – value at program point before n
- out_n – value at program point after n
- $f_n \in$ **F** – transfer function for n (given in_n, computes out_n)
- Signature of in_n, out_n, f_n : **L** $\rightarrow$ **L**

Requires that solution satisfies

- $\forall n.$ $out_n = f_n(in_n)$
- $\forall n \neq n_0.$ $in_n = \vee \{ out_m \, . \, m \text{ in } pred(n) \}$
- in_{n0} = **I**, summarizes information at the start of program

Dataflow Equations

Compiler processes program to obtain a set of dataflow equations

$$out_n := \quad f_n(in_n)$$

$$in_n \ := \ \vee \{ \ out_m \ . \text{ for each m in pred(n) } \}$$

Conceptually separates analysis problem from program

Worklist Algorithm for Solving Forward Dataflow Equations

```
for each n do out_n := f_n(⊥)

in_n0 := I; out_n0 := f_n0(I)
worklist := N - { n_0 }

while worklist ≠ ∅ do
      remove a node n from worklist
      in_n  := ∨ { out_m . m in pred(n) }
      out_n := f_n(in_n)
      if out_n changed then
            worklist := worklist ∪ succ(n)
```

Correctness Argument

Why does the result satisfy dataflow equations?

- Whenever it processes a node n, algorithm sets $\mathtt{out}_n \mathrel{:=} \mathtt{f}_n(\mathtt{in}_n)$
  Therefore, the algorithm ensures that $out_n = f_n(in_n)$

- Whenever out_m changes, it puts succ(m) on worklist. Consider any node n $\in$ succ(m). It will eventually come off worklist and algorithm will set

  $$\mathtt{in}_n \mathrel{:=} \vee \{\ \mathtt{out}_m\ .\ \mathtt{m\ in\ pred(n)}\ \}$$

  to ensure that $in_n = \vee \{ out_m . m \text{ in } pred(n) \}$

- So final solution will satisfy dataflow equations

- Need also to ensure that the dataflow equalities correspond to the states in the program execution (this comes later!)

Termination Argument

Why does algorithm terminate?

Sequence of values taken on by IN_n or OUT_n is a chain. If values stop increasing, worklist empties and algorithm terminates.

If lattice has <u>ascending chain property</u>, algorithm terminates

- **Algorithm terminates for finite lattices**
- For lattices with infinite length, use **widening operator**
  - Detect lattice values that may be part of infinitely ascending chain
  - Artificially raise value to least upper bound of chain

Termination Argument (Details)

- For finite lattice **(L, ≤)**
- Start: each node n ∈ CFG has an initial IN set, called $IN_0[n]$
- When F is **monotone,** for each n, successive values of IN[n] form a non-decreasing sequence.
  - Any chain starting at x ∈ L has at most c_x elements
  - x=IN[n] can increase in value at most c_x times
  - Then $C = \max_{n \in CFG} c_{IN[n]}$ is finite
- On every iteration, at least one IN[.] set must increase in value
  - If loop executes N × C times, all IN[.] sets would be T
  - The algorithm terminates in **O(N × C)** steps
    (but this is conservative)

Speed of Convergence

How quickly does the transfer function stabilize over backedge?

If the lattice has ascending chain property, then $\forall f \in F, \forall x \in L$ $f^{[k]}$ stabilizes, where

$$f^{[k]} = \bigwedge_{i=0..k} f^i(x) \quad where\ f^0 = x, f^i = f \circ f^{i-1}(x)$$

F is bounded if for all *f*, the chain $\{f^{[k]}\}$ is finite, k, bounded if k≥length

K-boundness: $f^k \geq f^{[k]}$ (if L has height k, then F will be k-bounded)

Fast: (2-bounded) $f \circ f \geq f \wedge x$

Rapid (1-semibound): $\forall f \in F, \forall x, y \in L\,.\ f(x) \leq y \wedge x \wedge f(y)$
which ends up being $\forall f \in F, \forall x \in L\,.\ x \leq f(x) \wedge f(\top)$

Speed of Convergence

Loop Connectedness d(G): for a reducible CFG G, it is the maximum number of back edges in any acyclic path in G.

Kam & Ullman, 1976:

- The depth-first version of the iterative algorithm halts in at most d(G) + 3 passes over the graph
- If the lattice L has $\top$, at most d(G) + 2 passes are needed

In practice:

- d(G) < 3, so the algorithm makes less than 6 passes over the graph

For mode details, see also Properties of data flow frameworks, Marlowe and Ryder (1990)

General Worklist Algorithm

(Reminder)

```
for each n do out_n := f_n(⊥)

in_n0 := I; out_n0 := f_n0(I)
worklist := N - { n_0 }

while worklist ≠ ∅ do
      remove a node n from worklist
      in_n  := ∨ { out_m . m in pred(n) }
      out_n := f_n(in_n)
      if out_n changed then
            worklist := worklist ∪ succ(n)
```

Reaching Definitions Algorithm

(Reminder)

```
for all nodes n in N
   OUT[n] = emptyset; // OUT[n] = GEN[n];
IN[Entry] = emptyset;
OUT[Entry] = GEN[Entry];
Changed = N - { Entry }; // N = all nodes in graph

while (Changed != emptyset)
      choose a node n in Changed;
      Changed = Changed - { n };

      IN[n] = emptyset;
      for all nodes p in predecessors(n)
      IN[n] = IN[n] U OUT[p];

      OUT[n] = GEN[n] U (IN[n] - KILL[n]);

      if (OUT[n] changed)
         for all nodes s in successors(n)
         Changed = Changed U { s };
```

Reaching Definitions

```
for all nodes n in N
    OUT[n] = emptyset;
IN[Entry] = emptyset;
OUT[Entry] = GEN[Entry];
Changed = N - { Entry };

while (Changed != emptyset)
  choose a node n in Changed;
  Changed = Changed - { n };

  IN[n] = emptyset;
  for all nodes p in predecessors(n)
    IN[n] = IN[n] U OUT[p];

  OUT[n] = GEN[n] U (IN[n] - KILL[n]);

  if (OUT[n] changed)
    for all nodes s in succ(n)
       Changed = Changed U { s };
```

General Worklist

for each n do $out_n := f_n(\bot)$

$in_{n0} := I;\ out_{n0} := f_{n0}(I)$

worklist := N - { n_0 }

while worklist $\neq \varnothing$ do

  remove a node n from worklist

  $in_n := \vee \{ out_m \ .\ m \text{ in } pred(n) \}$

  $out_n := f_n(in_n)$

  if out_n changed then

    worklist := worklist $\cup$ succ(n)

Reaching Definitions

P = powerset of set of all definitions in program (all subsets of set of definitions in program)

$\vee = \cup$ (order is $\subseteq$)

$\perp = \varnothing$

$I = in_{n0} = \perp$

F = all functions f of the form $f(x) = a \cup (x-b)$

- b is set of definitions that node kills
- a is set of definitions that node generates

General pattern for many transfer functions

- $f(x) = GEN \cup (x\text{-}KILL)$

Does Reaching Definitions Framework Satisfy Properties?

$\subseteq$ satisfies conditions for $\leq$

- **Reflexivity:** $x \subseteq x$
- **Antisymmetry:** $x \subseteq y$ and $y \subseteq x$ implies $y = x$
- **Transitivity:** $x \subseteq y$ and $y \subseteq z$ implies $x \subseteq z$

F satisfies transfer function conditions

- **Identity:** $\lambda x.\emptyset \cup (x - \emptyset) = \lambda x.x \in F$
- **Distributivity:** Will show $f(x \cup y) = f(x) \cup f(y)$

$$
\begin{aligned}
f(x) \cup f(y) &= (a \cup (x - b)) \cup (a \cup (y - b)) \\
&= a \cup (x - b) \cup (y - b) = a \cup ((x \cup y) - b) \\
&= f(x \cup y)
\end{aligned}
$$

Does Reaching Definitions Framework Satisfy Properties?

What about composition of F?

Given $f_1(x) = a_1 \cup (x\text{-}b_1)$ and $f_2(x) = a_2 \cup (x\text{-}b_2)$
we must show $f_1(f_2(x))$ can be expressed as $a \cup (x - b)$

$$\begin{aligned} f_1(f_2(x)) &= a_1 \cup ((a_2 \cup (x\text{-}b_2)) - b_1) \\ &= a_1 \cup ((a_2 - b_1) \cup ((x\text{-}b_2) - b_1)) \\ &= (a_1 \cup (a_2 - b_1)) \cup ((x\text{-}b_2) - b_1)) \\ &= (a_1 \cup (a_2 - b_1)) \cup (x\text{-}(b_2 \cup b_1)) \end{aligned}$$

- Let $a = (a_1 \cup (a_2 - b_1))$ and $b = b_2 \cup b_1$
- Then $f_1(f_2(x)) = a \cup (x - b)$

General Result

All GEN/KILL transfer function frameworks satisfy the three properties:

- Identity
- Distributivity
- Composition

And all of them converge rapidly