Program Analysis (70020)

Monotone Frameworks

Herbert Wiklicky

Department of Computing
Imperial College London

herbert@doc.ic.ac.uk
h.wiklicky@imperial.ac.uk

Autumn 2024

Partially Ordered Set

A partial ordering is a relation on a set L, i.e.

$$\sqsubseteq : L \times L \to \{\mathbf{tt}, \mathbf{ff}\} \quad \text{or} \quad \sqsubseteq \;\subseteq\; L \times L$$

that is:

Partially Ordered Set

A partial ordering is a relation on a set L, i.e.

$$\sqsubseteq : L \times L \to \{\mathbf{tt}, \mathbf{ff}\} \quad \text{or} \quad \sqsubseteq \ \subseteq \ L \times L$$

that is:

- **reflexive** $\forall l : l \sqsubseteq l$,

Partially Ordered Set

A partial ordering is a relation on a set L, i.e.

$$\sqsubseteq : L \times L \to \{\mathbf{tt}, \mathbf{ff}\} \quad \text{or} \quad \sqsubseteq \ \subseteq \ L \times L$$

that is:

- **reflexive** $\forall l : l \sqsubseteq l$,
- **transitive** $\forall l_1, l_2, l_3 : l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$, and

Partially Ordered Set

A partial ordering is a relation on a set L, i.e.

$$\sqsubseteq: L \times L \to \{\mathbf{tt}, \mathbf{ff}\} \quad \text{or} \quad \sqsubseteq \;\subseteq\; L \times L$$

that is:

- **reflexive** $\forall l : l \sqsubseteq l$,
- **transitive** $\forall l_1, l_2, l_3 : l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$, and
- **anti-symmetric** $\forall l_1, l_2 : l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2$.

Partially Ordered Set

A partial ordering is a relation on a set L, i.e.

$$\sqsubseteq: L \times L \to \{\mathbf{tt}, \mathbf{ff}\} \quad \text{or} \quad \sqsubseteq \; \subseteq \; L \times L$$

that is:

- **reflexive** $\forall l : l \sqsubseteq l$,
- **transitive** $\forall l_1, l_2, l_3 : l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$, and
- **anti-symmetric** $\forall l_1, l_2 : l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2$.

A partially ordered set $(L, \sqsubseteq)$ is a set L equipped with a partial ordering $\sqsubseteq$ (sometimes written $\sqsubseteq_L$). We shall write $l_2 \sqsupseteq l_1$ for $l_1 \sqsubseteq l_2$ and $l_1 \sqsubset l_2$ for $l_1 \sqsubseteq l_2 \wedge l_1 \neq l_2$.

Examples of POS's

Example: Integers
The integers $\mathbf{Z}$ ordered in the usual way, i.e. for $i_1, i_2 \in \mathbf{Z}$:

$$i_1 \sqsubseteq i_2 \text{ iff } i_1 \leq i_2$$

Examples of POS's

Example: Integers
The integers $\mathbf{Z}$ ordered in the usual way, i.e. for $i_1, i_2 \in \mathbf{Z}$:

$$i_1 \sqsubseteq i_2 \text{ iff } i_1 \leq i_2$$

Example: Power-Set
Take a (finite) set X and consider at the set of all sub-sets of X, i.e. its power set $\mathcal{P}(X)$. A partial ordering on $\mathcal{P}(X)$ is given by inclusion, i.e. for two sub-sets $S_1, S_2 \in \mathcal{P}(X)$:

$$S_1 \sqsubseteq S_2 \text{ iff } S_1 \subseteq S_2$$

Upper/Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$.

A subset Y of L has $l \in L$ as an upper bound if

$$\forall l' \in Y : l' \sqsubseteq l$$

Upper/Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$.

A subset Y of L has $l \in L$ as an upper bound if

$$\forall l' \in Y : l' \sqsubseteq l$$

and as a lower bound if

$$\forall l' \in Y : l' \sqsupseteq l.$$

Least Upper/Greatest Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$ and $Y \subseteq L$.

A least upper bound l of Y is an upper bound of Y that satisfies $l \sqsubseteq l_0$ whenever l_0 is another upper bound of Y;

Least Upper/Greatest Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$ and $Y \subseteq L$.

A least upper bound l of Y is an upper bound of Y that satisfies $l \sqsubseteq l_0$ whenever l_0 is another upper bound of Y;

Similarly, a greatest lower bound l of Y is a lower bound of Y satisfying: $l_0 \sqsubseteq l$ whenever l_0 is another lower bound of Y.

Least Upper/Greatest Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$ and $Y \subseteq L$.

A least upper bound l of Y is an upper bound of Y that satisfies $l \sqsubseteq l_0$ whenever l_0 is another upper bound of Y;

Similarly, a greatest lower bound l of Y is a lower bound of Y satisfying: $l_0 \sqsubseteq l$ whenever l_0 is another lower bound of Y.

Note that subsets Y of a partially ordered set L need not have least upper bounds nor greatest lower bounds but when they exist they are unique (since $\sqsubseteq$ is anti-symmetric) and they are denoted $\bigsqcup Y$ and $\bigsqcap Y$, respectively.

Least Upper/Greatest Lower Bounds

Given a partially ordered set $(L, \sqsubseteq)$ and $Y \subseteq L$.

A least upper bound l of Y is an upper bound of Y that satisfies $l \sqsubseteq l_0$ whenever l_0 is another upper bound of Y;

Similarly, a greatest lower bound l of Y is a lower bound of Y satisfying: $l_0 \sqsubseteq l$ whenever l_0 is another lower bound of Y.

Note that subsets Y of a partially ordered set L need not have least upper bounds nor greatest lower bounds but when they exist they are unique (since $\sqsubseteq$ is anti-symmetric) and they are denoted $\bigsqcup Y$ and $\bigsqcap Y$, respectively.

Sometimes $\bigsqcup$ is called the *join operator* and $\bigsqcap$ the *meet operator* and we shall write $l_1 \sqcup l_2$ for $\bigsqcup\{l_1, l_2\}$ and similarly $l_1 \sqcap l_2$ for $\bigsqcap\{l_1, l_2\}$.

Complete Lattice

A *complete lattice*

$$L = (L, \sqsubseteq) = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$$

is a partially ordered set $(L, \sqsubseteq)$ such that all subsets have least upper bounds as well as greatest lower bounds.

Complete Lattice

A *complete lattice*

$$L = (L, \sqsubseteq) = (L, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$$

is a partially ordered set $(L, \sqsubseteq)$ such that all subsets have least upper bounds as well as greatest lower bounds.

Furthermore, define $\perp = \bigsqcup \emptyset = \bigsqcap L$ is the least element and $\top = \bigsqcap \emptyset = \bigsqcup L$ is the greatest element.

Power-Set Lattice

Take a (finite) set X and look again at its power set $\mathcal{P}(X)$. A partial ordering '$\sqsubseteq$' on $\mathcal{P}(X)$ is given as above by inclusion '$\subseteq$'.

Power-Set Lattice

Take a (finite) set X and look again at its power set $\mathcal{P}(X)$. A partial ordering '$\sqsubseteq$' on $\mathcal{P}(X)$ is given as above by inclusion '$\subseteq$'.

The *meet* and *join* operators are given by (set) *intersection*

$$S_1 \sqcap S_2 = S_1 \cap S_2$$

and (set) *union*

$$S_1 \sqcup S_2 = S_1 \cup S_2.$$

Power-Set Lattice

Take a (finite) set X and look again at its power set $\mathcal{P}(X)$. A partial ordering '$\sqsubseteq$' on $\mathcal{P}(X)$ is given as above by inclusion '$\subseteq$'.

The *meet* and *join* operators are given by (set) *intersection*

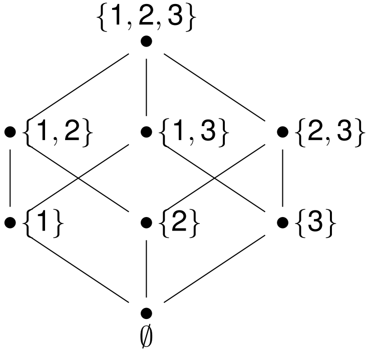
$$S_1 \sqcap S_2 = S_1 \cap S_2$$

and (set) *union*

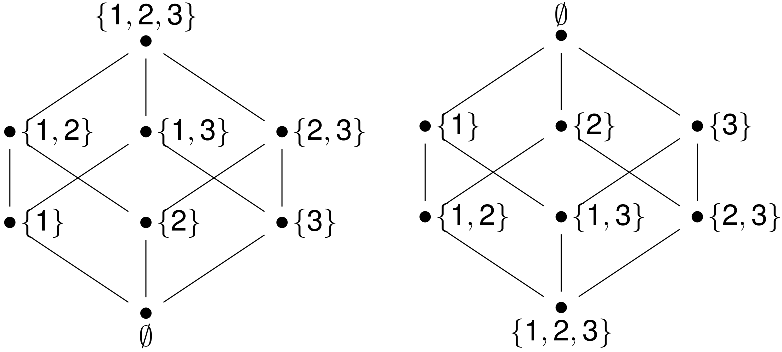
$$S_1 \sqcup S_2 = S_1 \cup S_2.$$

The least and greatest elements in $\mathcal{P}(X)$ are given by $\bot = \emptyset$ and $\top = X$.

Power-Set: Hasse Diagrams



Power-Set: Hasse Diagrams



Properties of Functions I

A function $f : L_1 \to L_2$ between two partially ordered sets $L_1 = (L_1, \sqsubseteq_1)$ and $L_2 = (L_2, \sqsubseteq_2)$ is *monotone* (or *isotone* or *order-preserving*) if

$$\forall l, l' \in L_1 : l \sqsubseteq_1 l' \quad \Rightarrow \quad f(l) \sqsubseteq_2 f(l')$$

Properties of Functions I

A function $f : L_1 \to L_2$ between two partially ordered sets $L_1 = (L_1, \sqsubseteq_1)$ and $L_2 = (L_2, \sqsubseteq_2)$ is *monotone* (or *isotone* or *order-preserving*) if

$$\forall l, l' \in L_1 : l \sqsubseteq_1 l' \quad \Rightarrow \quad f(l) \sqsubseteq_2 f(l')$$

A function $f : L_1 \to L_2$ is an *additive* function (or a *join morphism*, sometimes called a *distributive* function) if

$$\forall l_1, l_2 \in L_1 : f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

Properties of Functions I

A function $f : L_1 \to L_2$ between two partially ordered sets $L_1 = (L_1, \sqsubseteq_1)$ and $L_2 = (L_2, \sqsubseteq_2)$ is *monotone* (or *isotone* or *order-preserving*) if

$$\forall l, l' \in L_1 : l \sqsubseteq_1 l' \quad \Rightarrow \quad f(l) \sqsubseteq_2 f(l')$$

A function $f : L_1 \to L_2$ is an *additive* function (or a *join morphism*, sometimes called a *distributive* function) if

$$\forall l_1, l_2 \in L_1 : f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

and it is called a *multiplicative* function (or a *meet morphism*) if

$$\forall l_1, l_2 \in L_1 : f(l_1 \sqcap l_2) = f(l_1) \sqcap f(l_2)$$

Properties of Functions II

The function $f : L_1 \to L_2$ is a *completely additive* function (or a *complete join morphism*) if for all $Y \subseteq L_1$:

$$f\left(\bigsqcup_1 Y\right) = \bigsqcup_2 \{f(l') \mid l' \in Y\} \text{ whenever } \bigsqcup_1 Y \text{ exists}$$

Properties of Functions II

The function $f : L_1 \to L_2$ is a *completely additive* function (or a *complete join morphism*) if for all $Y \subseteq L_1$:

$$f\left(\bigsqcup_1 Y\right) = \bigsqcup_2 \{f(l') \mid l' \in Y\} \text{ whenever } \bigsqcup_1 Y \text{ exists}$$

and it is *completely multiplicative* (or a *complete meet morphism*) if for all $Y \subseteq L_1$:

$$f\left(\bigsqcap_1 Y\right) = \bigsqcap_2 \{f(l') \mid l' \in Y\} \text{ whenever } \bigsqcap_1 Y \text{ exists}$$

Cartesian Product $L_1 \times L_2$

Let $L_1 = (L_1, \sqsubseteq_1)$ and $L_2 = (L_2, \sqsubseteq_2)$ be partially ordered sets.
Define $L = (L, \sqsubseteq)$ by

$$L = L_1 \times L_2 = \{(l_1, l_2) \mid l_1 \in L_1 \wedge l_2 \in L_2\}$$

$$(l_{11}, l_{21}) \sqsubseteq (l_{12}, l_{22}) \ \text{ iff } \ l_{11} \sqsubseteq_1 l_{12} \ \wedge \ l_{21} \sqsubseteq_2 l_{22}$$

Cartesian Product $L_1 \times L_2$

Let $L_1 = (L_1, \sqsubseteq_1)$ and $L_2 = (L_2, \sqsubseteq_2)$ be partially ordered sets. Define $L = (L, \sqsubseteq)$ by

$$L = L_1 \times L_2 = \{(l_1, l_2) \mid l_1 \in L_1 \wedge l_2 \in L_2\}$$

$$(l_{11}, l_{21}) \sqsubseteq (l_{12}, l_{22}) \text{ iff } l_{11} \sqsubseteq_1 l_{12} \ \wedge \ l_{21} \sqsubseteq_2 l_{22}$$

If additionally each $L_i = (L_i, \sqsubseteq_i, \bigsqcup_i, \bigsqcap_i, \perp_i, \top_i)$ is a complete lattice then so is $L = (L, \sqsubseteq, \bigsqcup, \bigsqcap, \perp, \top)$ and furthermore

$$\bigsqcup Y = (\ \bigsqcup_1 \{l_1 \mid \exists\, l_2 : (l_1, l_2) \in Y\}\ ,\ \bigsqcup_2 \{l_2 \mid \exists\, l_1 : (l_1, l_2) \in Y\}\ )$$

and $\perp = (\perp_1, \perp_2)$ and similarly for $\bigsqcap Y$ and $\top$.

Total Function Space $S \to L_1$

Let $L_1 = (L_1, \sqsubseteq_1)$ be a partially ordered set and let S be a set.
Define $L = (L, \sqsubseteq)$ by

$$L = \{f : S \to L_1 \mid f \text{ is a total function}\}$$

$$f \sqsubseteq f' \ \text{ iff } \ \forall s \in S : f(s) \sqsubseteq_1 f'(s)$$

Total Function Space $S \to L_1$

Let $L_1 = (L_1, \sqsubseteq_1)$ be a partially ordered set and let S be a set.
Define $L = (L, \sqsubseteq)$ by

$$L = \{f : S \to L_1 \mid f \text{ is a total function}\}$$

$$f \sqsubseteq f' \ \text{ iff } \ \forall s \in S : f(s) \sqsubseteq_1 f'(s)$$

If additionally $L_1 = (L_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1)$ is a complete lattice then so is $L = (L, \sqsubseteq, \bigsqcup, \bigsqcap, \bot, \top)$ and furthermore

$$\bigsqcup Y = \lambda s. \bigsqcup{}_1 \{f(s) \mid f \in Y\}$$

and $\bot = \lambda s.\bot_1$ and similarly for $\bigsqcap Y$ and $\top$.

Chains

A subset $Y \subseteq L$ of a partially ordered set $L = (L, \sqsubseteq)$ is a *chain* if

$$\forall l_1, l_2 \in Y : (l_1 \sqsubseteq l_2) \vee (l_2 \sqsubseteq l_1)$$

Thus a chain is a (possibly empty) subset of L that is totally ordered.

We shall say that it is a *finite chain* if it is a finite subset of L.

Ascending and Descending Chains

A sequence $(l_n)_n = (l_n)_{n \in \mathbf{N}}$ of elements in L is an *ascending chain* if

$$n \leq m \Rightarrow l_n \sqsubseteq l_m$$

Writing $(l_n)_n$ also for $\{l_n \mid n \in \mathbf{N}\}$ it is clear that an ascending chain also is a chain.

Ascending and Descending Chains

A sequence $(l_n)_n = (l_n)_{n \in \mathbf{N}}$ of elements in L is an *ascending chain* if

$$n \leq m \Rightarrow l_n \sqsubseteq l_m$$

Writing $(l_n)_n$ also for $\{l_n \mid n \in \mathbf{N}\}$ it is clear that an ascending chain also is a chain.

Similarly, a sequence $(l_n)_n$ is a *descending chain* if

$$n \leq m \Rightarrow l_n \sqsupseteq l_m$$

Stabilising Chains

We shall say that a sequence $(I_n)_n$ *eventually stabilises* if and only if

$$\exists n_0 \in \mathbf{N} : \forall n \in \mathbf{N} : n \geq n_0 \Rightarrow I_n = I_{n_0}$$

Stabilising Chains

We shall say that a sequence $(l_n)_n$ *eventually stabilises* if and only if

$$\exists n_0 \in \mathbf{N} : \forall n \in \mathbf{N} : n \geq n_0 \Rightarrow l_n = l_{n_0}$$

For the sequence $(l_n)_n$ we write $\bigsqcup_n l_n$ for $\bigsqcup\{l_n \mid n \in \mathbf{N}\}$ and similarly we write $\bigsqcap_n l_n$ for $\bigsqcap\{l_n \mid n \in \mathbf{N}\}$.

ACC & DCC

We shall say that a partially ordered set $L = (L, \sqsubseteq)$ has *finite height* if and only if all chains are finite.

ACC & DCC

We shall say that a partially ordered set $L = (L, \sqsubseteq)$ has *finite height* if and only if all chains are finite.

It has finite height *at most* h if all chains contain at most $h + 1$ elements; it has finite height h if additionally there is a chain with $h + 1$ elements.

ACC & DCC

We shall say that a partially ordered set $L = (L, \sqsubseteq)$ has *finite height* if and only if all chains are finite.

It has finite height *at most* h if all chains contain at most $h + 1$ elements; it has finite height h if additionally there is a chain with $h + 1$ elements.

A partially ordered set L satisfies the *Ascending Chain Condition* (ACC) if and only if all ascending chains eventually stabilise.

ACC & DCC

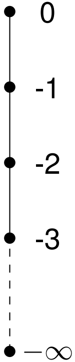
We shall say that a partially ordered set $L = (L, \sqsubseteq)$ has *finite height* if and only if all chains are finite.

It has finite height *at most* h if all chains contain at most $h + 1$ elements; it has finite height h if additionally there is a chain with $h + 1$ elements.

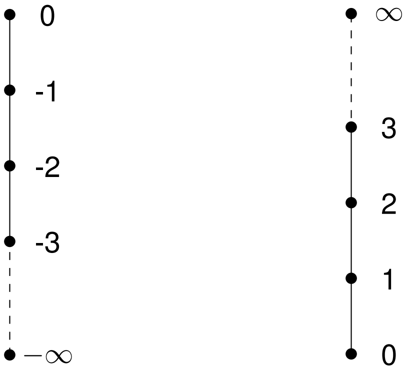
A partially ordered set L satisfies the *Ascending Chain Condition* (ACC) if and only if all ascending chains eventually stabilise.

A partially ordered set L satisfies the *Descending Chain Condition* (DCC) if and only if all descending chains eventually stabilise.

Chain Examples



Chain Examples



Reductive and Extensive Functions

Consider a monotone function $f : L \rightarrow L$ on a complete lattice L.

Reductive and Extensive Functions

Consider a monotone function $f : L \to L$ on a complete lattice L.

A *fixed point* of f is an element $l \in L$ such that $f(l) = l$, we write

$$Fix(f) = \{l \mid f(l) = l\}$$

for the set of fixed points.

Reductive and Extensive Functions

Consider a monotone function $f : L \to L$ on a complete lattice L.

A *fixed point* of f is an element $l \in L$ such that $f(l) = l$, we write

$$Fix(f) = \{l \mid f(l) = l\}$$

for the set of fixed points.

The function f is *reductive at* l if and only if $f(l) \sqsubseteq l$ and we write

$$Red(f) = \{l \mid f(l) \sqsubseteq l\}$$

for the set of elements upon which f is reductive; we shall say that f itself is *reductive* if $Red(f) = L$.

Reductive and Extensive Functions

Consider a monotone function $f : L \to L$ on a complete lattice L.

A *fixed point* of f is an element $l \in L$ such that $f(l) = l$, we write

$$Fix(f) = \{l \mid f(l) = l\}$$

for the set of fixed points.

The function f is *reductive at* l if and only if $f(l) \sqsubseteq l$ and we write

$$Red(f) = \{l \mid f(l) \sqsubseteq l\}$$

for the set of elements upon which f is reductive; we shall say that f itself is *reductive* if $Red(f) = L$. Similarly, the function f is *extensive at* l if and only if $f(l) \sqsupseteq l$, we write

$$Ext(f) = \{l \mid f(l) \sqsupseteq l\}$$

Fixed Points

Since L is a complete lattice it is always the case that the set $Fix(f)$ will have a greatest lower bound in L and we denote it by $lfp(f)$:

$$lfp(f) = \bigsqcap Fix(f) = \bigsqcap Red(f) \in Fix(f) \subseteq Red(f)$$

Fixed Points

Since L is a complete lattice it is always the case that the set $Fix(f)$ will have a greatest lower bound in L and we denote it by $lfp(f)$:

$$lfp(f) = \bigsqcap Fix(f) = \bigsqcap Red(f) \in Fix(f) \subseteq Red(f)$$

Similarly, the set $Fix(f)$ will have a least upper bound in L and we denote it by $gfp(f)$:

$$gfp(f) = \bigsqcup Fix(f) = \bigsqcup Ext(f) \in Fix(f) \subseteq Ext(f)$$

Existence of Fixed Points

If L satisfies the Ascending Chain Condition then there exists n such that $f^n(\perp) = f^{n+1}(\perp)$ and hence

$$lfp(f) = f^n(\perp).$$

Existence of Fixed Points

If L satisfies the Ascending Chain Condition then there exists n such that $f^n(\bot) = f^{n+1}(\bot)$ and hence

$$lfp(f) = f^n(\bot).$$

If L satisfies the Descending Chain Condition then there exists n such that $f^n(\top) = f^{n+1}(\top)$ and hence

$$gfp(f) = f^n(\top).$$

Existence of Fixed Points

If L satisfies the Ascending Chain Condition then there exists n such that $f^n(\bot) = f^{n+1}(\bot)$ and hence

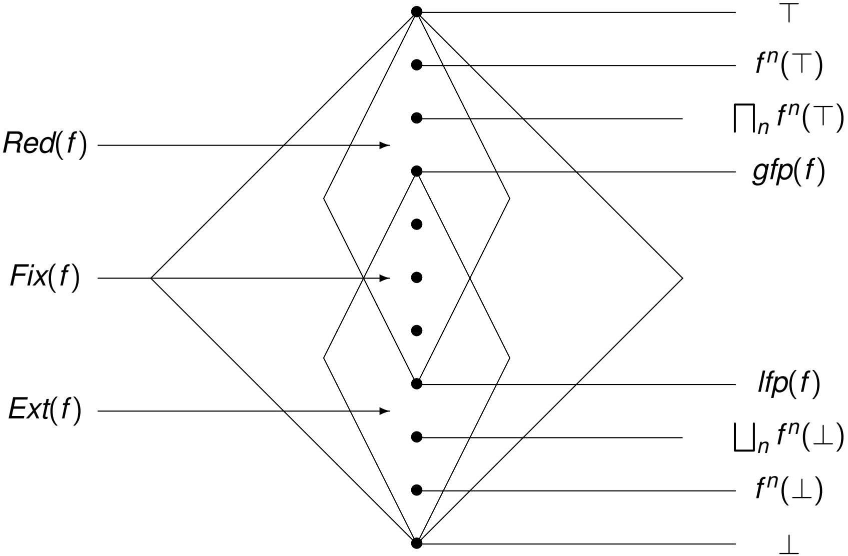
$$lfp(f) = f^n(\bot).$$

If L satisfies the Descending Chain Condition then there exists n such that $f^n(\top) = f^{n+1}(\top)$ and hence

$$gfp(f) = f^n(\top).$$

Indeed any monotone function f over a partially ordered set satisfying the Ascending Chain Condition is continuous.

Fix-points etc.



Fixed Points and Solutions

Given equations over some domain, e.g. integers

$$6x^3 - 3x^2 - x = 7$$

We look at it as a "*recursive*" equation:

$$6x^3 - 3x^2 - 7 = x$$

or simply:

$$f(x) = x.$$

Fixed Points and Solutions

Given equations over some domain, e.g. integers

$$6x^3 - 3x^2 - x = 7$$

We look at it as a "*recursive*" equation:

$$6x^3 - 3x^2 - 7 = x$$

or simply:

$$f(x) = x.$$

If x is a *fixed point* of f then it is a *solution* to the equation.

Lattice Equations

Given a system of equations with unknowns $x_1, \ldots, x_n$ over a complete lattice L (fulfilling ACC/DCC).

$$\begin{array}{rcl} x_1 & = & f_1(x_1, \ldots, x_n) \\ \ldots & & \ldots \\ x_n & = & f_m(x_1, \ldots, x_n) \end{array}$$

Lattice Equations

Given a system of equations with unknowns $x_1, \ldots, x_n$ over a complete lattice L (fulfilling ACC/DCC).

$$\begin{array}{rcl} x_1 & = & f_1(x_1, \ldots, x_n) \\ \ldots & & \ldots \\ x_n & = & f_m(x_1, \ldots, x_n) \end{array}$$

Consider the equations as defining a function $F : L^n \to L^n$

$$F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$$

Lattice Equations

Given a system of equations with unknowns $x_1, \ldots, x_n$ over a complete lattice L (fulfilling ACC/DCC).

$$\begin{array}{rcl} x_1 & = & f_1(x_1, \ldots, x_n) \\ \ldots & & \ldots \\ x_n & = & f_m(x_1, \ldots, x_n) \end{array}$$

Consider the equations as defining a function $F : L^n \to L^n$

$$F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$$

In our case we start with a recursive set of equations:

$$\text{Analysis}(i) = f_i(\text{Analysis}(1), \ldots, \text{Analysis}(n)).$$

Chaotic Iteration

Iteration: Construct iteratively the smallest or largest solution/fixed point, i.e. $\mathit{lfp}(F)$ or $\mathit{gfp}(F)$, by starting with

$$x_i = x_i^0 = \bot \quad \text{or} \quad x_i = x_i^0 = \top$$

and construct a sequence of approximations like:

Chaotic Iteration

Iteration: Construct iteratively the smallest or largest solution/fixed point, i.e. $lfp(F)$ or $gfp(F)$, by starting with

$$x_i = x_i^0 = \bot \quad \text{or} \quad x_i = x_i^0 = \top$$

and construct a sequence of approximations like:

$$\begin{array}{rcl} x_i^0 & = & \bot \\ x_i^1 & = & f(x_1^0, \ldots, x_n^0) \\ \ldots & & \ldots \\ x_i^k & = & f(x_1^{k-1}, \ldots, x_n^{k-1}) \end{array}$$

until we converge, i.e. the sequence stabilises.

An Example

Look at the complete lattice $\mathcal{P}(X) = \mathcal{P}(\{a, b, c, d\})$.
Construct solutions to the following set equations:

$$\begin{aligned}
S_1 &= \{a\} \cup S_4 \\
S_2 &= S_1 \cup S_3 \\
S_3 &= S_4 \cap \{b\} \\
S_4 &= S_2 \cup \{b, c\}
\end{aligned}$$

Two Solutions

Starting from $\perp$ gives:

$$\begin{array}{l|c|c|c|c|l}
S_1 = \emptyset & \{a\} & \{a, b, c\} & \{a, b, c\} & \{a, b, c\} & \ldots \\
S_2 = \emptyset & \emptyset & \{a\} & \{a, b, c\} & \{a, b, c\} & \ldots \\
S_3 = \emptyset & \emptyset & \{b\} & \{b\} & \{b\} & \ldots \\
S_4 = \emptyset & \{b, c\} & \{b, c\} & \{a, b, c\} & \{a, b, c\} & \ldots
\end{array}$$

Two Solutions

Starting from $\perp$ gives:

$$\begin{array}{l|c|c|c|c|l}
S_1 = \emptyset & \{a\} & \{a,b,c\} & \{a,b,c\} & \{a,b,c\} & \dots \\
S_2 = \emptyset & \emptyset & \{a\} & \{a,b,c\} & \{a,b,c\} & \dots \\
S_3 = \emptyset & \emptyset & \{b\} & \{b\} & \{b\} & \dots \\
S_4 = \emptyset & \{b,c\} & \{b,c\} & \{a,b,c\} & \{a,b,c\} & \dots
\end{array}$$

Starting from $\top$ gives:

$$\begin{array}{l|c|c|l}
S_1 = \{a,b,c,d\} & \{a,b,c,d\} & \{a,b,c,d\} & \dots \\
S_2 = \{a,b,c,d\} & \{a,b,c,d\} & \{a,b,c,d\} & \dots \\
S_3 = \{a,b,c,d\} & \{b\} & \{b\} & \dots \\
S_4 = \{a,b,c,d\} & \{a,b,c,d\} & \{a,b,c,d\} & \dots
\end{array}$$

Knaster-Tarski Fixed Point Theorem

Mathematics literature is full of Fixed Point Theorems, e.g.

Knaster-Tarski Fixed Point Theorem

Mathematics literature is full of Fixed Point Theorems, e.g.

Theorem (Knaster-Tarski)

Let L be a complete lattice and assume that $f : L \mapsto L$ is an order-preserving map. Then

$$\bigsqcup\{x \in L \mid x \sqsubseteq f(x)\} \in \mathit{Fix}(f).$$

Knaster-Tarski Fixed Point Theorem

Mathematics literature is full of Fixed Point Theorems, e.g.

Theorem (Knaster-Tarski)

Let L be a complete lattice and assume that $f : L \mapsto L$ is an order-preserving map. Then

$$\bigsqcup\{x \in L \mid x \sqsubseteq f(x)\} \in Fix(f).$$

B.A. Davey and H.A. Priestley: *Introduction to Lattices and Order*, Cambridge 1990.

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$
\begin{aligned}
\mathit{Analysis}_\circ(\ell) &= \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases} \\
\mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell))
\end{aligned}
$$

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$\begin{aligned}
\mathit{Analysis}_\circ(\ell) &= \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases} \\
\mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell))
\end{aligned}$$

$\bigsqcup$ is $\bigcap$ or $\bigcup$ (and $\sqcup$ is $\cup$ or $\cap$),

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$\begin{array}{rcl} \mathit{Analysis}_\circ(\ell) & = & \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases} \\ \mathit{Analysis}_\bullet(\ell) & = & f_\ell(\mathit{Analysis}_\circ(\ell)) \end{array}$$

F is either $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$,

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$
\begin{aligned}
\mathit{Analysis}_\circ(\ell) &= \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases} \\
\mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell))
\end{aligned}
$$

E is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$,

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$
\begin{array}{rcl}
\mathit{Analysis}_\circ(\ell) & = & \left\{ \begin{array}{l} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{array} \right. \\
\mathit{Analysis}_\bullet(\ell) & = & f_\ell(\mathit{Analysis}_\circ(\ell))
\end{array}
$$

ι specifies the initial or final analysis information, and

Classical Analyses

Each of the four classical analyses considers equations for a label consistent program $S_\star$ and they take the form:

$$
\begin{array}{rcl}
\mathit{Analysis}_\circ(\ell) & = & \begin{cases} \iota, \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\}, \text{otherwise} \end{cases} \\
\mathit{Analysis}_\bullet(\ell) & = & f_\ell(\mathit{Analysis}_\circ(\ell))
\end{array}
$$

f_ℓ is the transfer function associated with $B^\ell \in \mathit{blocks}(S_\star)$.

Forward vs Backward Analysis

The forward analyses have F to be $\mathit{flow}(S_\star)$ and then $\mathit{Analysis}_\circ$ concerns entry conditions and $\mathit{Analysis}_\bullet$ concerns exit conditions; also the equation system presupposes that $S_\star$ has isolated entries.

Forward vs Backward Analysis

The forward analyses have F to be $\mathit{flow}(S_\star)$ and then $\mathit{Analysis}_\circ$ concerns entry conditions and $\mathit{Analysis}_\bullet$ concerns exit conditions; also the equation system presupposes that $S_\star$ has isolated entries.

The backward analyses have F to be $\mathit{flow}^R(S_\star)$ and then $\mathit{Analysis}_\circ$ concerns exit conditions and $\mathit{Analysis}_\bullet$ concerns entry conditions; also the equation system presupposes that $S_\star$ has isolated exits.

Must vs May Analysis

When $\bigsqcup$ is $\bigcap$ we require the *greatest* sets that solve the equations and we are able to detect properties satisfied by *all* paths of execution reaching (or leaving) the entry (or exit) of a label; these analyses are often called must analyses.

Must vs May Analysis

When $\bigsqcup$ is $\bigcap$ we require the *greatest* sets that solve the equations and we are able to detect properties satisfied by *all* paths of execution reaching (or leaving) the entry (or exit) of a label; these analyses are often called must analyses.

When $\bigsqcup$ is $\bigcup$ we require the *least* sets that solve the equations and we are able to detect properties satisfied by *at least one* execution path to (or from) the entry (or exit) of a label; these analyses are often called may analyses.

Alternative Formulation

It is occasionally awkward to have to assume that forward analyses have isolated entries and that backward analyses have isolated exits. This motivates reformulating the above equations to be of the form:

$$\begin{aligned} \mathit{Analysis}_\circ(\ell) &= \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\} \sqcup \iota_E^\ell \\ \mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell)) \end{aligned}$$

Alternative Formulation

It is occasionally awkward to have to assume that forward analyses have isolated entries and that backward analyses have isolated exits. This motivates reformulating the above equations to be of the form:

$$\begin{aligned} \mathit{Analysis}_\circ(\ell) &= \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\} \sqcup \iota_E^\ell \\ \mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell)) \end{aligned}$$

where

$$\iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases}$$

and $\perp$ satisfies $l \sqcup \perp = l$ (hence $\perp$ is not really there).

Transfer Functions

The view that we take here is that a program is a *transition system*; the nodes represent blocks and each block has a transfer function associated with it that specifies how the block acts on the “input” state.

Transfer Functions

The view that we take here is that a program is a *transition system*; the nodes represent blocks and each block has a transfer function associated with it that specifies how the block acts on the “input” state.

Note that for forward analyses, the input state is the entry state, and for backward analyses, it is the exit state.

Monotone & Distributive Frameworks

A Monotone Framework consists of:

Monotone & Distributive Frameworks

A Monotone Framework consists of:

- a complete lattice, L, that satisfies the Ascending Chain Condition, and we write $\bigsqcup$ for the least upper bound operator; and

Monotone & Distributive Frameworks

A Monotone Framework consists of:

- a complete lattice, L, that satisfies the Ascending Chain Condition, and we write $\bigsqcup$ for the least upper bound operator; and
- a set $\mathcal{F}$ of monotone functions from L to L that contains the identity function and that is closed under function composition.

Monotone & Distributive Frameworks

A Monotone Framework consists of:

- a complete lattice, L, that satisfies the Ascending Chain Condition, and we write $\bigsqcup$ for the least upper bound operator; and
- a set $\mathcal{F}$ of monotone functions from L to L that contains the identity function and that is closed under function composition.

A Distributive Framework is a Monotone Framework where additionally all functions f in $\mathcal{F}$ are required to be distributive:

$$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;
- the space of transfer functions, $\mathcal{F}$, of the framework;

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;
- the space of transfer functions, $\mathcal{F}$, of the framework;
- a finite flow, F, that typically is $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$;

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;
- the space of transfer functions, $\mathcal{F}$, of the framework;
- a finite flow, F, that typically is $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$;
- a finite set of so-called extremal labels, E, that typically is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$;

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;
- the space of transfer functions, $\mathcal{F}$, of the framework;
- a finite flow, F, that typically is $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$;
- a finite set of so-called extremal labels, E, that typically is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$;
- an extremal value, $\iota \in L$, for the extremal labels; and

Instance of a Framework

An *instance*, Analysis, of a Monotone or Distributive Framework to consists of:

- the complete lattice, L, of the framework;
- the space of transfer functions, $\mathcal{F}$, of the framework;
- a finite flow, F, that typically is $flow(S_\star)$ or $flow^R(S_\star)$;
- a finite set of so-called extremal labels, E, that typically is $\{init(S_\star)\}$ or $final(S_\star)$;
- an extremal value, $\iota \in L$, for the extremal labels; and
- a mapping, $f_\cdot$, from the labels $\mathbf{Lab}_\star$ of F to transfer functions in $\mathcal{F}$.

Equations

An instance gives rise to a *set of equations*, Analysis$^=$, of the form considered earlier:

$$
\begin{array}{rcl}
\mathit{Analysis}_\circ(\ell) & = & \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota_E^\ell \\
 & & \text{where } \iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases} \\
\mathit{Analysis}_\bullet(\ell) & = & f_\ell(\mathit{Analysis}_\circ(\ell))
\end{array}
$$

Classical Instances

	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
L	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\bigsqcup$	$\bigcap$	$\bigcup$	$\bigcap$	$\bigcup$
$\bot$	$\mathbf{AExp}_\star$	$\emptyset$	$\mathbf{AExp}_\star$	$\emptyset$
ι	$\emptyset$	$\{(x,?) \mid x \in FV(S_\star)\}$	$\emptyset$	$\emptyset$
E	$\{\mathit{init}(S_\star)\}$	$\{\mathit{init}(S_\star)\}$	$\mathit{final}(S_\star)$	$\mathit{final}(S_\star)$
F	$\mathit{flow}(S_\star)$	$\mathit{flow}(S_\star)$	$\mathit{flow}^R(S_\star)$	$\mathit{flow}^R(S_\star)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
f_ℓ	$f_\ell(l) = (l \setminus \mathit{kill}([B]^\ell)) \cup \mathit{gen}([B]^\ell)$ where $[B]^\ell \in \mathit{blocks}(S_\star)$			

Classical Monotone Frameworks

Lemma: Each of the four classical data flow analyses is a Monotone Framework as well as a Distributive Framework.

Classical Monotone Frameworks

Lemma: Each of the four classical data flow analyses is a Monotone Framework as well as a Distributive Framework.

It is worth pointing out that in order to get this result we have made the frameworks dependent upon the actual program – this is needed to enforce that the Ascending Chain Condition is fulfilled.

A Non-Distributive Example

The Constant Propagation Analysis (CP) will determine:

For each program point, whether or not a variable has a constant value whenever execution reaches that point.

A Non-Distributive Example

The Constant Propagation Analysis (CP) will determine:

> *For each program point, whether or not a variable has a constant value whenever execution reaches that point.*

Such information can be used as the basis for an optimisation known as *Constant Folding*: all uses of the variable may be replaced by the constant value.

CP State: $\mathbf{Z}^{\top}$

The (abstract) states for the CP Analysis are given by:

$$\widehat{\textbf{State}}_{\mathsf{CP}} = ((\textbf{Var}_{\star} \to \mathbf{Z}^{\top})_{\perp}, \sqsubseteq, \sqcup, \sqcap, \perp, \lambda x.\top)$$

where $\textbf{Var}_{\star}$ is the set of variables appearing in the program.

CP State: $\mathbf{Z}^\top$

The (abstract) states for the CP Analysis are given by:

$$\widehat{\mathbf{State}}_{\mathrm{CP}} = ((\mathbf{Var}_\star \to \mathbf{Z}^\top)_\perp, \sqsubseteq, \sqcup, \sqcap, \perp, \lambda x.\top)$$

where $\mathbf{Var}_\star$ is the set of variables appearing in the program.

$\mathbf{Z}^\top = \mathbf{Z} \cup \{\top\}$ is partially ordered as follows:

$$\forall z \in \mathbf{Z}^\top : z \sqsubseteq \top$$

$$\forall z_1, z_2 \in \mathbf{Z} : (z_1 \sqsubseteq z_2) \Leftrightarrow (z_1 = z_2)$$

CP State: Partial Order $\mathbf{Z}^\top$ vs Lattice $\mathbf{Z}^\top_\perp$

We construct a non-standard partial order on $\mathbf{Z}$:

$$\cdots \quad -2 \quad -1 \quad 0 \quad +1 \quad +2 \quad \cdots$$

CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}_{\perp}^{\top}$

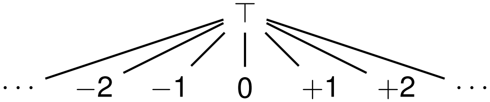
We construct a non-standard partial order on $\mathbf{Z}$:

$$\top$$

$$\cdots \quad -2 \quad -1 \quad 0 \quad +1 \quad +2 \quad \cdots$$

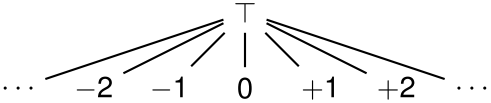
CP State: Partial Order $\mathbf{Z}^\top$ vs Lattice $\mathbf{Z}_\perp^\top$

We construct a non-standard partial order on $\mathbf{Z}$:



CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}^{\top}_{\perp}$

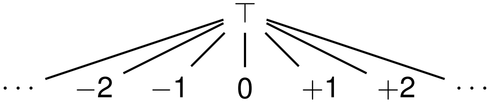
We construct a non-standard partial order on $\mathbf{Z}$:



We could also consider a complete lattice $\mathbf{Z}^{\top}_{\perp}$ (but don't):

CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}_{\perp}^{\top}$

We construct a non-standard partial order on $\mathbf{Z}$:

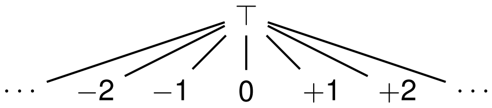


We could also consider a complete lattice $\mathbf{Z}_{\perp}^{\top}$ (but don't):

$$\cdots \quad -2 \quad -1 \quad 0 \quad +1 \quad +2 \quad \cdots$$

CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}^{\top}_{\perp}$

We construct a non-standard partial order on $\mathbf{Z}$:



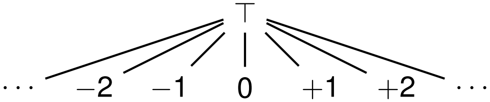
We could also consider a complete lattice $\mathbf{Z}^{\top}_{\perp}$ (but don't):

$$\top$$

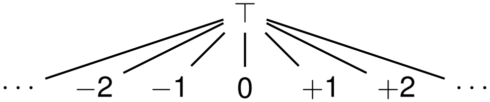
$$\cdots \quad -2 \quad -1 \quad 0 \quad +1 \quad +2 \quad \cdots$$

CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}^{\top}_{\bot}$

We construct a non-standard partial order on $\mathbf{Z}$:

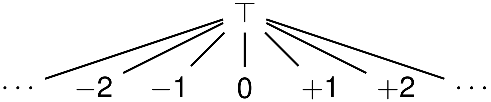


We could also consider a complete lattice $\mathbf{Z}^{\top}_{\bot}$ (but don't):

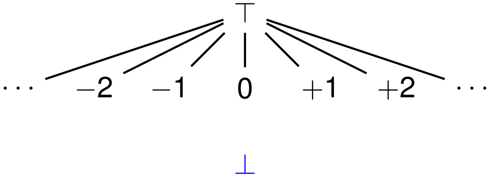


CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}^{\top}_{\bot}$

We construct a non-standard partial order on $\mathbf{Z}$:

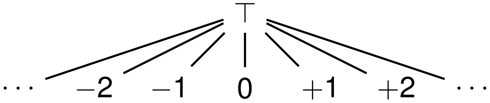


We could also consider a complete lattice $\mathbf{Z}^{\top}_{\bot}$ (but don't):

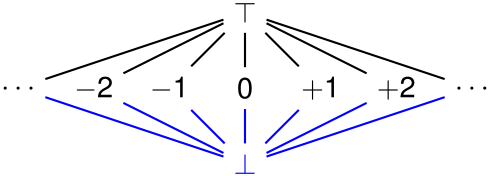


CP State: Partial Order $\mathbf{Z}^{\top}$ vs Lattice $\mathbf{Z}_{\perp}^{\top}$

We construct a non-standard partial order on $\mathbf{Z}$:



We could also consider a complete lattice $\mathbf{Z}_{\perp}^{\top}$ (but don't):



CP State: Lattice

To capture the case where no information is available we extend $\mathbf{Var}_\star \to \mathbf{Z}^\top$ with a least element $\bot$, written $(\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot$.

CP State: Lattice

To capture the case where no information is available we extend $\mathbf{Var}_\star \to \mathbf{Z}^\top$ with a least element $\bot$, written $(\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot$.

The partial ordering $\sqsubseteq$ on $\widehat{\mathbf{State}}_{\mathsf{CP}} = (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot$ is:

$$\forall \widehat{\sigma} \in (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot : \quad \bot \sqsubseteq \widehat{\sigma}$$

$$\forall \widehat{\sigma}_1, \widehat{\sigma}_2 \in \mathbf{Var}_\star \to \mathbf{Z}^\top : \quad \widehat{\sigma}_1 \sqsubseteq \widehat{\sigma}_2 \;\text{ iff }\; \forall x : \widehat{\sigma}_1(x) \sqsubseteq \widehat{\sigma}_2(x)$$

CP State: Lattice

To capture the case where no information is available we extend $\mathbf{Var}_\star \to \mathbf{Z}^\top$ with a least element $\bot$, written $(\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot$.

The partial ordering $\sqsubseteq$ on $\widehat{\mathbf{State}}_{\mathrm{CP}} = (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot$ is:

$$\forall \widehat{\sigma} \in (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot : \quad \bot \sqsubseteq \widehat{\sigma}$$

$$\forall \widehat{\sigma}_1, \widehat{\sigma}_2 \in \mathbf{Var}_\star \to \mathbf{Z}^\top : \quad \widehat{\sigma}_1 \sqsubseteq \widehat{\sigma}_2 \ \text{ iff } \ \forall x : \widehat{\sigma}_1(x) \sqsubseteq \widehat{\sigma}_2(x)$$

and the binary least upper bound operation is then:

$$\forall \widehat{\sigma} \in (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot : \quad \widehat{\sigma} \sqcup \bot = \widehat{\sigma} = \bot \sqcup \widehat{\sigma}$$

$$\forall \widehat{\sigma}_1, \widehat{\sigma}_2 \in \mathbf{Var}_\star \to \mathbf{Z}^\top : \quad \forall x : (\widehat{\sigma}_1 \sqcup \widehat{\sigma}_2)(x) = \widehat{\sigma}_1(x) \sqcup \widehat{\sigma}_2(x)$$

CP State Evaluation

$$\mathcal{A}_{\mathsf{CP}} \;:\; \textbf{AExp} \to (\widehat{\textbf{State}}_{\mathsf{CP}} \to \textbf{Z}_{\bot}^{\top})$$

CP State Evaluation

$$\mathcal{A}_{\mathrm{CP}} \;:\; \mathbf{AExp} \to (\widehat{\mathbf{State}}_{\mathrm{CP}} \to \mathbf{Z}_{\bot}^{\top})$$

$$\mathcal{A}_{\mathrm{CP}}[\![x]\!]\widehat{\sigma} = \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ \widehat{\sigma}(x) & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathrm{CP}}[\![n]\!]\widehat{\sigma} = \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ n & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathrm{CP}}[\![a_1 \; op_a \; a_2]\!]\widehat{\sigma} = \mathcal{A}_{\mathrm{CP}}[\![a_1]\!]\widehat{\sigma} \; \widehat{\mathrm{op}}_a \; \mathcal{A}_{\mathrm{CP}}[\![a_2]\!]\widehat{\sigma}$$

CP State Evaluation

$$
\begin{array}{rcl}
\mathcal{A}_{\mathrm{CP}} & : & \mathbf{AExp} \to (\widehat{\mathbf{State}_{\mathrm{CP}}} \to \mathbf{Z}_{\perp}^{\top}) \\
\mathcal{A}_{\mathrm{CP}}[\![x]\!]\widehat{\sigma} & = & \begin{cases} \perp & \text{if } \widehat{\sigma} = \perp \\ \widehat{\sigma}(x) & \text{otherwise} \end{cases} \\
\mathcal{A}_{\mathrm{CP}}[\![n]\!]\widehat{\sigma} & = & \begin{cases} \perp & \text{if } \widehat{\sigma} = \perp \\ n & \text{otherwise} \end{cases} \\
\mathcal{A}_{\mathrm{CP}}[\![a_1 \ op_a \ a_2]\!]\widehat{\sigma} & = & \mathcal{A}_{\mathrm{CP}}[\![a_1]\!]\widehat{\sigma} \ \widehat{\mathrm{op}}_a \ \mathcal{A}_{\mathrm{CP}}[\![a_2]\!]\widehat{\sigma}
\end{array}
$$

The operations on $\mathbf{Z}$ are lifted to $\mathbf{Z}_{\perp}^{\top} = \mathbf{Z} \cup \{\perp, \top\}$ by taking $z_1 \ \widehat{\mathrm{op}}_a \ z_2 = z_1 \ \mathbf{op}_a \ z_2$ if $z_1, z_2 \in \mathbf{Z}$ (and where $\mathbf{op}_a$ is the corresponding arithmetic operation on $\mathbf{Z}$), $z_1 \ \widehat{\mathrm{op}}_a \ z_2 = \perp$ if $z_1 = \perp$ or $z_2 = \perp$ and $z_1 \ \widehat{\mathrm{op}}_a \ z_2 = \top$ otherwise.

CP Transfer Function

$$\mathcal{F}_{\mathsf{CP}} = \{f \mid f \text{ is a monotone function on } \widehat{\textbf{State}}_{\mathsf{CP}}\}$$

CP Transfer Function

$$\mathcal{F}_{\mathsf{CP}} = \{f \mid f \text{ is a monotone function on } \widehat{\textbf{State}}_{\mathsf{CP}}\}$$

$$\begin{array}{lrcl}
[x := a]^{\ell}: & f_{\ell}^{\mathsf{CP}}(\widehat{\sigma}) & = & \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ \widehat{\sigma}[x \mapsto \mathcal{A}_{\mathsf{CP}}[\![a]\!]\widehat{\sigma}] & \text{otherwise} \end{cases} \\
[\textbf{skip}]^{\ell}: & f_{\ell}^{\mathsf{CP}}(\widehat{\sigma}) & = & \widehat{\sigma} \\
[b]^{\ell}: & f_{\ell}^{\mathsf{CP}}(\widehat{\sigma}) & = & \widehat{\sigma}
\end{array}$$

CP Flow

Constant Propagation (CP) is a forward analysis, so for the program $S_\star$ we take the flow, F, to be $\mathit{flow}(S_\star)$.

The extremal labels, E, are given by $\{\mathit{init}(S_\star)\}$, and the extremal value, ι_{CP}, is $\lambda x.\top$. The property lattice L and transfer function $\mathcal{F}_{\mathsf{CP}}$ as above.

CP Flow

Constant Propagation (CP) is a forward analysis, so for the program $S_\star$ we take the flow, F, to be $\mathit{flow}(S_\star)$.

The extremal labels, E, are given by $\{\mathit{init}(S_\star)\}$, and the extremal value, ι_{CP}, is $\lambda x.\top$. The property lattice L and transfer function $\mathcal{F}_{\mathsf{CP}}$ as above.

Lemma: Constant Propagation is a Monotone Framework that is *not* a Distributive Framework.

Distributive Framework

To show that it is not a Distributive Framework consider the transfer function f_ℓ^{CP} for $[y := x * x]^\ell$ and let $\widehat{\sigma}_1$ and $\widehat{\sigma}_2$ be such that $\widehat{\sigma}_1(x) = 1$ and $\widehat{\sigma}_2(x) = -1$.

Distributive Framework

To show that it is not a Distributive Framework consider the transfer function f_ℓ^{CP} for $[y := x * x]^\ell$ and let $\widehat{\sigma}_1$ and $\widehat{\sigma}_2$ be such that $\widehat{\sigma}_1(x) = 1$ and $\widehat{\sigma}_2(x) = -1$.

Then $\widehat{\sigma}_1 \sqcup \widehat{\sigma}_2$ maps x to $\top$ and thus $f_\ell^{\mathsf{CP}}(\widehat{\sigma}_1 \sqcup \widehat{\sigma}_2)$ maps y to $\top$ and hence fails to record that y has the constant value 1.

Distributive Framework

To show that it is not a Distributive Framework consider the transfer function f_ℓ^{CP} for $[y := x * x]^\ell$ and let $\widehat{\sigma}_1$ and $\widehat{\sigma}_2$ be such that $\widehat{\sigma}_1(x) = 1$ and $\widehat{\sigma}_2(x) = -1$.

Then $\widehat{\sigma}_1 \sqcup \widehat{\sigma}_2$ maps x to $\top$ and thus $f_\ell^{\mathrm{CP}}(\widehat{\sigma}_1 \sqcup \widehat{\sigma}_2)$ maps y to $\top$ and hence fails to record that y has the constant value 1.

However, both $f_\ell^{\mathrm{CP}}(\widehat{\sigma}_1)$ and $f_\ell^{\mathrm{CP}}(\widehat{\sigma}_2)$ map y to 1 and so does $f_\ell^{\mathrm{CP}}(\widehat{\sigma}_1) \sqcup f_\ell^{\mathrm{CP}}(\widehat{\sigma}_2)$.

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^\ell$.

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^{\ell}$.

x	y
-1	$\top$

x	y
$+1$	1

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^{\ell}$.

x	y
-1	$\top$

$\sqcup$

x	y
$\top$	$\top$

x	y
$+1$	1

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^{\ell}$.

x	y
-1	$\top$

x	y
$+1$	1

$\sqcup$

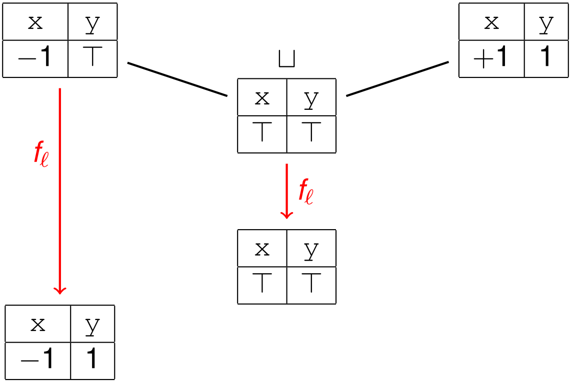
x	y
$\top$	$\top$

$$\downarrow f_{\ell}$$

x	y
$\top$	$\top$

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^{\ell}$.



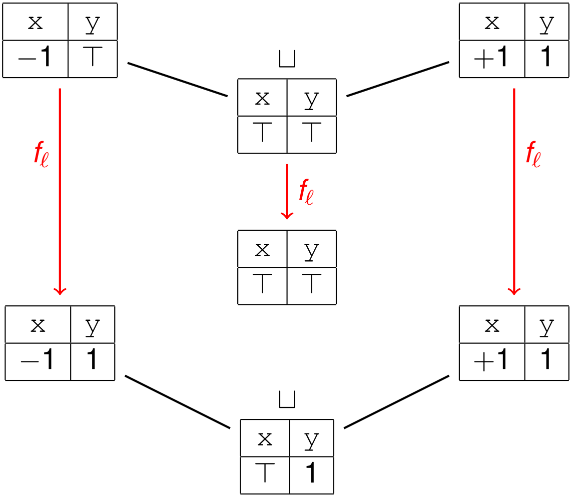
Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^\ell$.

$$\begin{tikzcd}
\begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline -1 & \top \\ \hline\end{array} \arrow[r, dash] \arrow[dd, "f_\ell"'] & \overset{\sqcup}{\begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline \top & \top \\ \hline\end{array}} \arrow[d, "f_\ell"] & \begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline +1 & 1 \\ \hline\end{array} \arrow[l, dash] \arrow[dd, "f_\ell"] \\
& \begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline \top & \top \\ \hline\end{array} & \\
\begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline -1 & 1 \\ \hline\end{array} & & \begin{array}{|c|c|}\hline \mathtt{x} & \mathtt{y} \\ \hline +1 & 1 \\ \hline\end{array}
\end{tikzcd}$$

Transfer Functions and Information Collection

Consider transfer function for the block $[y := x * x]^{\ell}$.



The MFP Solution (1)

INPUT: An instance of a Monotone Framework: $(L, \mathcal{F}, F, E, \iota, f_\cdot)$

OUTPUT: $MFP_\circ, MFP_\bullet$

The MFP Solution (1)

INPUT: An instance of a Monotone Framework: $(L, \mathcal{F}, F, E, \iota, f_\cdot)$

OUTPUT: $MFP_\circ, MFP_\bullet$

Step 1: Initialisation (of W and Analysis)

W := nil;
for all (ℓ, ℓ') in F do
    W := cons((ℓ, ℓ'),W);
for all ℓ in F or E do
    if $\ell \in E$ then Analysis[ℓ] := ι
        else Analysis[ℓ] := $\perp_L$;

The MFP Solution (2&3)

Step 2: Iteration (updating W and Analysis)

while W $\neq$ nil do

ℓ := fst(head(W)); ℓ' = snd(head(W));

W := tail(W);

if f_ℓ(Analysis[ℓ]) $\not\sqsubseteq$ Analysis[ℓ'] then

Analysis[ℓ'] := Analysis[ℓ'] $\sqcup$ f_ℓ(Analysis[ℓ]);

for all (ℓ', ℓ'') in F do W := cons((ℓ', ℓ''),W);

The MFP Solution (2&3)

Step 2: Iteration (updating W and Analysis)

while W $\neq$ nil do

ℓ := fst(head(W)); ℓ' = snd(head(W));

W := tail(W);

if f_ℓ(Analysis[ℓ]) $\not\sqsubseteq$ Analysis[ℓ'] then

Analysis[ℓ'] := Analysis[ℓ'] $\sqcup$ f_ℓ(Analysis[ℓ]);

for all (ℓ', ℓ'') in F do W := cons((ℓ', ℓ''),W);

Step 3: Presenting the result ($MFP_\circ$ and $MFP_\bullet$)

for all ℓ in F or E do

$MFP_\circ(\ell)$:= Analysis[ℓ];

$MFP_\bullet(\ell)$:= f_ℓ(Analysis[ℓ])

MFP Termination

Given an instance of a Monotone Framework $(L, \mathcal{F}, F, E, \iota, f_{\cdot})$ with a property lattice L fullfilling the ACC/DCC.

Starting from $\perp$ and using iterative (approximation) methods like Chaotic Iteration or the Worklist Algorithm (which optimses the iterations by only considering updates when "necessary") we can compute solutions $Analysis_{\circ}$ and $Analysis_{\bullet}$.

MFP Termination

Given an instance of a Monotone Framework $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ with a property lattice L fullfilling the ACC/DCC.

Starting from $\perp$ and using iterative (approximation) methods like Chaotic Iteration or the Worklist Algorithm (which optimses the iterations by only considering updates when "necessary") we can compute solutions $\mathit{Analysis}_\circ$ and $\mathit{Analysis}_\bullet$.

Lemma: The iterative construction of a solution (using chaotic iteration, worklist algorithm) always terminates and it computes the least MFP solution (more precisely $\mathit{MFP}_\circ$ and $\mathit{MFP}_\bullet$) to the instance of the framework.

MFP Complexity

Assume that the flow F is represented in such a way that all (ℓ', ℓ'') emanating from ℓ' can be found in time proportional to their number. Suppose that E and F contain at most $b \geq 1$ distinct labels, that F contains at most $e \geq b$ pairs, and that L has finite height at most $h \geq 1$.

MFP Complexity

Assume that the flow F is represented in such a way that all (ℓ', ℓ'') emanating from ℓ' can be found in time proportional to their number. Suppose that E and F contain at most $b \geq 1$ distinct labels, that F contains at most $e \geq b$ pairs, and that L has finite height at most $h \geq 1$.

Then steps 1 and 3 perform at most $O(b + e)$ basic operations. In step 2 a pair is placed on the worklist at most $O(h)$ times, and each time it takes only a constant number of basic steps to process it; this yields at most $O(e \cdot h)$ basic operations for step 2. Since $h \geq 1$ and $e \geq b$ this gives at most $O(e \cdot h)$ basic operations for the algorithm.

RD Complexity

Consider the Reaching Definitions Analysis and suppose that there are at most $v \geq 1$ variables and $b \geq 1$ labels in the program, $S_\star$, being analysed. Since $L = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$, it follows that $h \leq v \cdot b$ and thus we have an $O(v \cdot b^3)$ upper bound on the number of basic operations.

RD Complexity

Consider the Reaching Definitions Analysis and suppose that there are at most $v \geq 1$ variables and $b \geq 1$ labels in the program, $S_\star$, being analysed. Since $L = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$, it follows that $h \leq v \cdot b$ and thus we have an $O(v \cdot b^3)$ upper bound on the number of basic operations.

Better: If $S_\star$ is label consistent then the variable of the pairs (x, ℓ) of $\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$ will always be uniquely determined by the label ℓ so we get an $O(b^3)$ upper bound on the number of basic operations. Furthermore, F is $\mathit{flow}(S_\star)$ and inspection of the equations for $\mathit{flow}(S_\star)$ shows that for each label ℓ we construct at most two pairs with ℓ in the first component. This means that $e \leq 2 \cdot b$ and we get an $O(b^2)$ upper bound on the number of basic operations.

MOP Solution: Paths

Consider an instance $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ of a Monotone Framework.

MOP Solution: Paths

Consider an instance $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ of a Monotone Framework.

We shall use the notation $\vec{\ell} = [\ell_1, \cdots, \ell_n]$ for a sequence of $n \geq 0$ labels.

MOP Solution: Paths

Consider an instance $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ of a Monotone Framework.

We shall use the notation $\vec{\ell} = [\ell_1, \cdots, \ell_n]$ for a sequence of $n \geq 0$ labels.

The paths up to *but not* including ℓ are:

$$path_\circ(\ell) = \{[\ell_1, \cdots, \ell_{n-1}] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

The paths up to *and* including ℓ are:

$$path_\bullet(\ell) = \{[\ell_1, \cdots, \ell_n] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

MOP Solutions

For a path $\vec{\ell} = [\ell_1, \cdots, \ell_n]$ we define the transfer function

$$f_{\vec{\ell}} = f_{\ell_n} \circ \cdots \circ f_{\ell_1} \circ id$$

so that for the empty path we have $f_{[\,]} = id$ where *id* is the identity function.

MOP Solutions

For a path $\vec{\ell} = [\ell_1, \cdots, \ell_n]$ we define the transfer function

$$f_{\vec{\ell}} = f_{\ell_n} \circ \cdots \circ f_{\ell_1} \circ id$$

so that for the empty path we have $f_{[\,]} = id$ where id is the identity function.

The MOP solutions are then given by:

$$MOP_{\circ}(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_{\circ}(\ell) \}$$

$$MOP_{\bullet}(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_{\bullet}(\ell) \}$$

MOP Solution: Termination

Unfortunately, the MOP solution sometimes cannot be computable (meaning that it is undecidable what the solution is) even though the MFP solution is always easily computable (because of the property space satisfying the Ascending Chain Condition); the following result establishes one such result:

MOP Solution: Termination

Unfortunately, the MOP solution sometimes cannot be computable (meaning that it is undecidable what the solution is) even though the MFP solution is always easily computable (because of the property space satisfying the Ascending Chain Condition); the following result establishes one such result:

Lemma: The MOP solution for the Constant Propagation Analysis is undecidable.

MFP and MOP Solutions

Lemma: Consider the MFP and the MOP solutions to an instance $(L, \mathcal{F}, F, B, \iota, f_\cdot)$ of a Monotone Framework; then:

$$MFP_\circ \sqsupseteq MOP_\circ \text{ and } MFP_\bullet \sqsupseteq MOP_\bullet$$

MFP and MOP Solutions

Lemma: Consider the MFP and the MOP solutions to an instance $(L, \mathcal{F}, F, B, \iota, f_\cdot)$ of a Monotone Framework; then:

$$MFP_\circ \sqsupseteq MOP_\circ \text{ and } MFP_\bullet \sqsupseteq MOP_\bullet$$

If the framework is a Distributive Framework and if $path_\circ(\ell) \neq \emptyset$ for all ℓ in E and F then:

$$MFP_\circ = MOP_\circ \text{ and } MFP_\bullet = MOP_\bullet$$

MFP and MOP Solutions

Lemma: Consider the MFP and the MOP solutions to an instance $(L, \mathcal{F}, F, B, \iota, f_\cdot)$ of a Monotone Framework; then:

$$MFP_\circ \sqsupseteq MOP_\circ \text{ and } MFP_\bullet \sqsupseteq MOP_\bullet$$

If the framework is a Distributive Framework and if $path_\circ(\ell) \neq \emptyset$ for all ℓ in E and F then:

$$MFP_\circ = MOP_\circ \text{ and } MFP_\bullet = MOP_\bullet$$

It is always possible to formulate the MOP solution as an MFP solution over a different property space (like $\mathcal{P}(L)$) and therefore little is lost by focusing on the fixed point approach to Monotone Frameworks.