• My mail: mgapb2@cam.ac.uk. Do not hesitate to ask questions!
• Course notes will be updated, keep an eye on the course webpage.
INTRODUCTION
• **Formal methods**: tools for the specification, development, analysis and verification of software and hardware systems.
What is this course about?

- **Formal methods**: tools for the specification, development, analysis and verification of software and hardware systems.
- **Programming language theory**: how to design, implement and reason about programming languages?
What is this course about?

- **Formal methods**: tools for the specification, development, analysis and verification of software and hardware systems.
- **Programming language theory**: how to design, implement and reason about programming languages?
- **Programming language semantics**: what is the (mathematical) meaning of a program?
What is this course about?

- Formal methods: tools for the specification, development, analysis and verification of software and hardware systems.
- Programming language theory: how to design, implement and reason about programming languages?
- Programming language semantics: what is the (mathematical) meaning of a program?

Goal: give an abstract and compositional (mathematical) model of programs.
Why should we care?

- **Insight**: exposes the mathematical “essence” of programming language concepts.
Why should we care?

- **Insight**: exposes the mathematical “essence” of programming language concepts.
- **Language design**: feedback from semantic concepts (monads, algebraic effects & effect handlers...).
Why should we care?

- **Insight**: exposes the mathematical “essence” of programming language concepts.
- **Language design**: feedback from semantic concepts (monads, algebraic effects & effect handlers...).
- **Rigour**: semantics is necessary to specify/justify formal methods (compilers, type systems, code analysis, certification...).
STYLES OF FORMAL SEMANTICS

- Operational
- Axiomatic
- Denotational
 STYLES OF FORMAL SEMANTICS

- **Operational**: meaning of a program in terms of the steps of computation it takes during execution (see Part IB Semantics).
- **Axiomatic**
- **Denotational**
Styles of Formal Semantics

- **Operational**: meaning of a program in terms of the steps of computation it takes during execution (see Part IB Semantics).
- **Axiomatic**: indirect meaning of a program in terms of a program logic to reason about its properties (see Part II Hoare Logic & Model Checking).
- **Denotational**
• **Operational**: meaning of a program in terms of the *steps of computation* it takes during execution (see Part IB Semantics).

• **Axiomatic**: indirect meaning of a program in terms of a *program logic* to reason about its properties (see Part II Hoare Logic & Model Checking).

• **Denotational**: meaning of a program defined abstractly as object of some suitable *mathematical structure* (see this course).
Denotational semantics in a nutshell

Syntax $\rightarrow$ Semantics

Program $P$ $\mapsto$ Denotation $[P]$

Recursive program $\mapsto$ Partial recursive function

Boolean circuit $\mapsto$ Boolean function

...
<table>
<thead>
<tr>
<th>Syntax</th>
<th>$\rightarrow$</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program $P$</td>
<td>$\mapsto$</td>
<td>Denotation $[P]$</td>
</tr>
</tbody>
</table>

- Recursive program $\mapsto$ Partial recursive function
- Boolean circuit $\mapsto$ Boolean function
- ...$\mapsto$ ...

<table>
<thead>
<tr>
<th>Type</th>
<th>$\mapsto$</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>$\mapsto$</td>
<td>Continuous functions between domains</td>
</tr>
</tbody>
</table>
Abstraction

- mathematical object, implementation/machine independent;
- captures the abstract essence of programming language concepts;
- should relate to practical implementations, though...
Properties of Denotational Semantics

Abstraction

- mathematical object, implementation/machine independent;
- captures the abstract essence of programming language concepts;
- should relate to practical implementations, though...

Compositionality

- The denotation of a phrase is defined using the *denotation* of its sub-phrases.
- $[P]$ represents the contribution of $P$ to *any* program containing $P$.
- Much more flexible than whole-program semantics.
INTRODUCTION

A BASIC EXAMPLE
Commands

\[ C \in \text{Comm} ::= \text{skip} \mid L := A \mid C ; C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \]
IMP syntax

Arithmetic expressions

\[ A \in \text{Exp} ::= n \mid L \mid A + A \mid \ldots \]

Boolean expressions

\[ B \in \text{Exp} ::= \text{true} \mid \text{false} \mid A = A \mid \neg B \mid \ldots \]

Commands

\[ C \in \text{Comm} ::= \text{skip} \mid L := A \mid C;C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \]

ranges over a set \( L \) of locations
Arithmetic expressions

\[ A \in \text{Aexp} ::= n \mid L \mid A + A \mid \ldots \]

Commands

\[ C \in \text{Comm} ::= \text{skip} \mid L ::= A \mid C;C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \]
IMP SYNTAX

Arithmetic expressions

\[ A \in \text{Aexp} ::= n \mid L \mid A + A \mid \ldots \]

Ranges over integers

Commands

\[ C \in \text{Comm} ::= \text{skip} \mid L := A \mid C;C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \]
Arithmetic expressions

\[ A \in \text{Aexp} ::= n \mid L \mid A + A \mid ... \]

Boolean expressions

\[ B \in \text{Bexp} ::= \text{true} \mid \text{false} \mid A = A \mid \neg B \mid ... \]

Commands

\[ C \in \text{Comm} ::= \text{skip} \mid L := A \mid C;C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \]
Denotation functions – naïvely

\[ A : \text{Aexp} \rightarrow \mathbb{Z} \]

where

\[ \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \]
Denotation functions – naïvely

\[ A : \text{Aexp} \rightarrow \mathbb{Z} \]
\[ B : \text{Bexp} \rightarrow \mathbb{B} \]

where

\[ \mathbb{Z} = \{..., -1, 0, 1,...\} \]
\[ \mathbb{B} = \{\text{true, false}\} \]
ARITHMETIC EXPRESSIONS?

\[ A[n] = n \]

ARITHMETIC EXPRESSIONS?

\[ A[n] = n \]


\[ A[L] = ??? \]
State = ($\mathbb{L} \rightarrow \mathbb{Z}$)

where $\mathbb{Z}$ = {…,−1,0,1,…} and $\mathbb{B}$ = {true,false}. 
Denotation functions

State = ($\mathbb{L} \rightarrow \mathbb{Z}$)

$A : Aexp \rightarrow (State \rightarrow \mathbb{Z})$

$B : Bexp \rightarrow (State \rightarrow \mathbb{B})$

where

$\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$

$\mathbb{B} = \{true, false\}$. 
Denotation functions

\[ \text{State} = (\mathbb{L} \to \mathbb{Z}) \]

\[ A : \text{Aexp} \to (\text{State} \to \mathbb{Z}) \]
\[ B : \text{Bexp} \to (\text{State} \to \mathbb{B}) \]
\[ C : \text{Comm} \to (\text{State} \to \text{State}) \]

where \( \rightarrow \) denotes partial functions and

\[ \mathbb{Z} = \{..., -1, 0, 1, ...\} \]
\[ \mathbb{B} = \{\text{true, false}\} \]
Semantics of arithmetic expressions

\[ A[n] = \lambda s \in \text{State}. n \]

\[ A[A_1 + A_2] = \lambda s \in \text{State}. A[A_1](s) + A[A_2](s) \]
Semantics of arithmetic expressions

\[ A[n] = \lambda s \in \text{State}. \, n \]

\[ A[A_1 + A_2] = \lambda s \in \text{State.} \, A[A_1](s) + A[A_2](s) \]

\[ A[L] = \lambda s \in \text{State.} \, s(L) \]
\[ B[\text{true}] = \lambda s \in \text{State}. \text{true} \]

\[ B[\text{false}] = \lambda s \in \text{State}. \text{false} \]

\[ B[A_1 = A_2] = \lambda s \in \text{State. } eq(A[A_1](s), A[A_2](s)) \]

where \( eq(a, a') = \begin{cases} 
  \text{true} & \text{if } a = a' \\
  \text{false} & \text{if } a \neq a'
\end{cases} \)
\[ C[\text{skip}] = \lambda s \in \text{State}. s \]
$C[\text{skip}] = \lambda s \in \text{State}. s$

$C[\text{if } B \text{ then } C \text{ else } C'] = \lambda s \in \text{State}. \text{if } (C[B](s), C[C](s), C[C'])(s))$

where $\text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases}$
**Semantics of commands**

\[
C[	ext{skip}] = \lambda s \in \text{State}. s
\]

\[
C[\text{if } B \text{ then } C \text{ else } C'] = \lambda s \in \text{State}. \text{if}(C[B](s), C[C](s), C[C'](s))
\]

where \(\text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases}\)

This is compositionality!
Semantics of commands

\[ C[\text{skip}] = \lambda s \in \text{State.}\ s \]

\[ C[\text{if } B \text{ then } C \text{ else } C'] = \lambda s \in \text{State.} \text{ if } (C[B](s), C[C](s), C[C'](s)) \]

where \( \text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases} \)

\[ C[L := A] = \lambda s \in \text{State.} s[L \mapsto A[A](s)] \]

where \( s[L \mapsto n](L') = \begin{cases} n & \text{if } L' = L \\ s(L) & \text{otherwise} \end{cases} \)
SEMANTICS OF COMMANDS

\[
C[\text{skip}] = \lambda s \in \text{State}. s
\]

\[
C[\text{if } B \text{ then } C \text{ else } C'] = \lambda s \in \text{State}. \text{if } (C[B](s), C[C](s), C[C'](s))
\]

where \( \text{if}(b, x, x') = \begin{cases} 
  x & \text{if } b = \text{true} \\
  x' & \text{if } b = \text{false} 
\end{cases} \)

\[
C[L := A] = \lambda s \in \text{State}. s[L \mapsto A[A](s)]
\]

where \( s[L \mapsto n](L') = \begin{cases} 
  n & \text{if } L' = L \\
  s(L) & \text{otherwise} 
\end{cases} \)

\[
C[C;C'] = C[C'] \circ C[C]
= \lambda s \in \text{State}. C[C'](C[C](s))
\]
INTRODUCTION

A SEMANTICS FOR LOOPS
This is all very nice, but...

\[ [\text{while } B \text{ do } C] = \ ??? \]
This is all very nice, but...

\[ [\text{while } B \text{ do } C] = ??? \]

Remember:

- \((\text{while } B \text{ do } C, s) \rightarrow (\text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip, } s)\)
- we want a *compositional* semantic: we should give \([\text{while } B \text{ do } C]\) in terms of \([C]\) and \([B]\)
\[
[\text{while } B \text{ do } C] = [\text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip}]
\]
\[
= \lambda s \in \text{State}. \text{if}(\llbracket B \rrbracket, \llbracket \text{while } B \text{ do } C \rrbracket \circ \llbracket C \rrbracket (s), s)
\]
LOOP AS A FIXPOINT

\[
\begin{align*}
[\text{while } B \text{ do } C] &= \left[ \text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip} \right] \\
&= \lambda s \in \text{State}. \text{if}(B, [\text{while } B \text{ do } C] \circ [C](s), s)
\end{align*}
\]

Not a direct definition for \([\text{while } B \text{ do } C]\)... But a **fixed point equation**!

\[
[\text{while } B \text{ do } C] = F_{[B],[C]}(\text{while } B \text{ do } C)
\]

where \(F_{b,c} : (\text{State} \rightarrow \text{State}) \rightarrow (\text{State} \rightarrow \text{State})\)

\[
\begin{align*}
\text{where } F_{b,c} & : (\text{State} \rightarrow \text{State}) \rightarrow (\text{State} \rightarrow \text{State}) \\
w & \mapsto \lambda s \in \text{State}. \text{if}(b, w \circ c(s), s).
\end{align*}
\]
Now we have a goal

• Why/when does $w = F_{b,c}(w)$ have a solution?
• What if it has several solutions? Which one should be our $\texttt{while B do C}$?
Now we have a goal

- Why/when does $w = F_{b,c}(w)$ have a solution?
- What if it has several solutions? Which one should be our `while B do C`?

Our occupation for the next few lectures...
INTRODUCTION

A TASTE OF DOMAIN THEORY
\[\text{while } X > 0 \text{ do (} Y := X \ast Y; X := X - 1)\]
An example

\[
\text{while } X > 0 \text{ do } (Y := X \ast Y; X := X - 1)
\]

should be some \( w \) such that:

\[
w = F_{[X > 0], [Y := X \ast Y; X := X - 1]}(w).
\]
\[ \text{while } X > 0 \text{ do } (Y := X \ast Y; X := X - 1) \]

should be some \( w \) such that:

\[ w = F_{[[X > 0], [Y := X \ast Y; X := X - 1]]}(w). \]

That is, we are looking for a fixed point of the following \( F : D \to D \), where \( D \) is (State \( \rightarrow \) State):

\[ F(w)([X \leftrightarrow x, Y \leftrightarrow y]) = \begin{cases} [X \leftrightarrow x, Y \leftrightarrow y] & \text{if } x \leq 0 \\ w([X \leftrightarrow x - 1, Y \leftrightarrow x \cdot y]) & \text{if } x > 0. \end{cases} \]
THE POSET OF PARTIAL FUNCTIONS

<table>
<thead>
<tr>
<th>Partial order $\sqsubseteq$ on $D (= \text{State} \rightarrow \text{State})$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w \sqsubseteq w'$ if</td>
</tr>
<tr>
<td>for all $s \in \text{State}$, if $w$ is defined at $s$</td>
</tr>
<tr>
<td>then so is $w'$ and moreover $w(s) = w'(s)$.</td>
</tr>
<tr>
<td>if the graph of $w$ is included in the graph of $w'$.</td>
</tr>
</tbody>
</table>
**THE POSET OF PARTIAL FUNCTIONS**

**Partial order \( \subseteq \) on \( D (= \text{State} \rightarrow \text{State}) \):**

- \( w \subseteq w' \) if for all \( s \in \text{State} \), if \( w \) is defined at \( s \) then so is \( w' \) and moreover \( w(s) = w'(s) \).
- if the graph of \( w \) is included in the graph of \( w' \).

**Least element \( \perp \in D \):**

- \( \perp = \) totally undefined partial function
- \( = \) partial function with empty graph
Define $\mathbf{w}_n = F^n(\mathbf{w})$, that is

$$\begin{cases} 
\mathbf{w}_0 &= \bot \\
\mathbf{w}_{n+1} &= F(\mathbf{w}_n) 
\end{cases}$$
Define $w_n = F^n(w)$, that is
\[
\begin{align*}
  w_0 &= \bot \\
  w_{n+1} &= F(w_n).
\end{align*}
\]

\[w_1[X \mapsto x, Y \mapsto y] = F(\bot)[X \mapsto x, Y \mapsto y] = \begin{cases} 
  [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
  \text{undefined} & \text{if } x \geq 1
\end{cases} \]
Define \( w_n = F^n(w) \), that is
\[
\begin{cases}
  w_0 &= \bot \\
  w_{n+1} &= F(w_n)
\end{cases}
\]

\[
 w_2[X \mapsto x, Y \mapsto y] = F(w_1)[X \mapsto x, Y \mapsto y] = \begin{cases}
  [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
  [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\
  \text{undefined} & \text{if } x \geq 2
\end{cases}
\]
Define $w_n = F^n(w)$, that is
$$
\begin{align*}
\begin{cases}
    w_0 & = \bot \\
    w_{n+1} & = F(w_n)
\end{cases}
\end{align*}
$$

$$
\begin{align*}
    w_3[X \mapsto x, Y \mapsto y] &= F(w_2)[X \mapsto x, Y \mapsto y] = \\
    &\begin{cases}
        [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
        [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\
        [X \mapsto 0, Y \mapsto 2y] & \text{if } x = 2 \\
        \text{undefined} & \text{if } x \geq 3
    \end{cases}
\end{align*}
$$
Define \( w_n = F^n(w) \), that is
\[
\begin{align*}
  w_0 &= \bot \\
  w_{n+1} &= F(w_n).
\end{align*}
\]

\( w_n[X \mapsto x, Y \mapsto y] = \begin{cases} 
  [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\
  [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\
  \text{undefined} & \text{if } x \geq n
\end{cases} \)
Approximating the Fixed Point

Define $w_n = F^n(w)$, that is

$$
\begin{align*}
  w_0 &= \perp \\
  w_{n+1} &= F(w_n).
\end{align*}
$$

$$
\begin{align*}
  w_n[X \mapsto x, Y \mapsto y] &= \begin{cases} 
    [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\
    [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\
    \text{undefined} & \text{if } x \geq n
  \end{cases}
\end{align*}
$$

$$
\begin{align*}
  w_0 \subseteq w_1 \subseteq \ldots \subseteq w_n \subseteq \ldots
\end{align*}
$$
Approximating the Fixed Point

Define $w_n = F^n(w)$, that is

$$
\begin{aligned}
    w_0 &= \bot \\
    w_{n+1} &= F(w_n).
\end{aligned}
$$

$$
\begin{aligned}
    w_n\{X \mapsto x, Y \mapsto y\} &= \\
    \begin{cases}
        [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\
        [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\
        \text{undefined} & \text{if } x \geq n
    \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
    w_0 \sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots \sqsubseteq w_\infty?
\end{aligned}
$$
APPROXIMATING THE FIXED POINT

Define $w_n = F^n(w)$, that is

$$
\begin{align*}
  w_0 &= \bot \\
  w_{n+1} &= F(w_n).
\end{align*}
$$

$$
 w_n[X \mapsto x, Y \mapsto y] = \begin{cases}
  [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\
  [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\
  \text{undefined} & \text{if } x \geq n
\end{cases}
$$

$$
 w_0 \sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots \sqsubseteq w_\infty
$$

$$
 w_\infty[X \mapsto x, Y \mapsto y] = \bigsqcup_{i \in \mathbb{N}} w_i = \begin{cases}
  [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\
  [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } x \geq 0
\end{cases}
$$
We have our semantics

\[ F(w_\infty)(X \mapsto x, Y \mapsto y) \]
\[ F(w_\infty)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x - 1, Y \mapsto x \cdot y] & \text{if } x > 0 \end{cases} \] (by definition of \( F \))
We have our semantics

\[
F(\omega_\infty)[X \leftrightarrow x, Y \leftrightarrow y] = \begin{cases} 
    [X \leftrightarrow x, Y \leftrightarrow y] & \text{if } x \leq 0 \\
    \omega_\infty[X \leftrightarrow x - 1, Y \leftrightarrow x \cdot y] & \text{if } x > 0 
\end{cases} 
\]

(by definition of \( F' \))

\[
= \begin{cases} 
    [X \leftrightarrow x, Y \leftrightarrow y] & \text{if } x \leq 0 \\
    [X \leftrightarrow 0, Y \leftrightarrow (x - 1)! \cdot x \cdot y] & \text{if } x > 0 
\end{cases} 
\]

(by definition of \( \omega_\infty \))
WE HAVE OUR SEMANTICS

\[ F(w_\infty)[X \mapsto x, Y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
\omega_\infty[X \mapsto x - 1, Y \mapsto x \cdot y] & \text{if } x > 0
\end{cases} \]  
(by definition of \( F \))

\[ = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
[X \mapsto 0, Y \mapsto (x - 1)! \cdot x \cdot y] & \text{if } x > 0
\end{cases} \]  
(by definition of \( \omega_\infty \))

\[ = \omega_\infty[X \mapsto x, Y \mapsto y] \]
WE HAVE OUR SEMANTICS

\[
F(w_\infty)[X \mapsto x, Y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
[w_\infty[X \mapsto x-1, Y \mapsto x \cdot y]] & \text{if } x > 0 
\end{cases} 
\]
(by definition of \(F\))

\[
= \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
[X \mapsto 0, Y \mapsto (x-1)! \cdot x \cdot y] & \text{if } x > 0 
\end{cases} 
\]
(by definition of \(w_\infty\))

\[
= w_\infty[X \mapsto x, Y \mapsto y] 
\]

\cdot \(w_\infty\) is a fixed point

\cdot which moreover agrees with the operational semantics (!)
LEAST FIXED POINTS
LEAST FIXED POINTS

POSETS AND MONOTONE FUNCTIONS
A partial order on a set $D$ is a binary relation $\sqsubseteq$ that is

- reflexive: $\forall d \in D. \, d \sqsubseteq d$
- transitive: $\forall d, d', d'' \in D. \, d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$
- anti-symmetric: $\forall d, d' \in D. \, d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.
A **partial order** on a set $D$ is a binary relation $\sqsubseteq$ that is

**reflexive**: $\forall d \in D. \; d \sqsubseteq d$

**transitive**: $\forall d, d', d'' \in D. \; d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$

**anti-symmetric**: $\forall d, d' \in D. \; d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.
**Domain of partial functions** $X \rightarrow Y$

**Underlying set:** partial functions $f$ with domain of definition $\text{dom}(f) \subseteq X$ and taking values in $Y$;
Domain of partial functions $X \rightarrow Y$

**Underlying set:** partial functions $f$ with domain of definition $\text{dom}(f) \subseteq X$ and taking values in $Y$;

**Order:** $f \subseteq g$ if $\text{dom}(f) \subseteq \text{dom}(g)$ and $\forall x \in \text{dom}(f). f(x) = g(x)$, i.e. if $\text{graph}(f) \subseteq \text{graph}(g)$. 
A function $f: D \to E$ between posets is **monotone** if

$$\forall d, d' \in D. \quad d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$
A function $f: D \rightarrow E$ between posets is **monotone** if

$$\forall d, d' \in D. \ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$
LEAST FIXED POINTS

LEAST ELEMENTS AND PRE-FIXED POINTS
An element $d \in S$ is the **least** element of $S$ if it satisfies

$$\forall x \in S. \, d \sqsubseteq x.$$
An element $d \in S$ is the least element of $S$ if it satisfies

$$\forall x \in S. \ d \sqsubseteq x.$$ 

If it exists, it is unique, and is written $\bot_S$, or simply $\bot$.

$$\begin{align*}
\text{LEAST} & \quad x \in S \\
\bot_S \sqsubseteq x
\end{align*}$$
An element $d \in S$ is the least element of $S$ if it satisfies

$$\forall x \in S. \ d \sqsubseteq x.$$ 

If it exists, it is unique, and is written $\perp_S$, or simply $\perp$. 

\[
\begin{align*}
\text{LEAST} & \quad \frac{x \in S}{\perp_S \sqsubseteq x} \\
\text{ASYM} & \quad \frac{\perp_S \sqsubseteq \perp_S}{\perp_S = \perp_S'} \quad \text{LEAST} \quad \frac{\perp_S' \in S}{\perp_S \sqsubseteq \perp_S'} \\
& \quad \text{LEAST} \quad \frac{\perp_S \sqsubseteq \perp_S}{\perp_S = \perp_S''}
\end{align*}
\]
An element $d \in D$ is a **pre-fixed point** of $f$ if it satisfies $f(d) \sqsubseteq d$. 

**LFP-FIX**

$f(fix(f)) \sqsubseteq fix(f)$

**LFP-LEAST**

$f(d) \sqsubseteq d$

$fix(f) \sqsubseteq d$
An element \( d \in D \) is a pre-fixed point of \( f \) if it satisfies \( f(d) \subseteq d \).

The least pre-fixed point of \( f \), if it exists, will be written

\[ \text{fix}(f) \]
An element $d \in D$ is a **pre-fixed point** of $f$ if it satisfies $f(d) \sqsubseteq d$.

The **least pre-fixed point** of $f$, if it exists, will be written

$$\text{fix}(f)$$

It is thus (uniquely) specified by the two properties:

-LFP-FIX: $f(\text{fix}(f)) \sqsubseteq \text{fix}(f)$

-LFP-LEAST: $f(d) \sqsubseteq d \Rightarrow \text{fix}(f) \sqsubseteq d$
The least pre-fixed point is a fixed point.
To prove $\text{fix}(f) \sqsubseteq d$, it is enough to show $f(d) \sqsubseteq d$. 
Proofs with least fixed points

\[ \text{LFP-FIX} \quad f(fix(f)) \sqsubseteq fix(f) \]
\[ \text{LFP-LEAST} \quad f(d) \sqsubseteq d \]
\[ \text{fix}(f) \sqsubseteq d \]

Application: least pre-fixed points of monotone functions are (least) fixed points.

\[ \text{LFP-FIX} \quad f(fix(f)) \sqsubseteq fix(f) \]
\[ \text{ASYM} \quad f(fix(f)) \sqsubseteq fix(f) \]
\[ \text{LFP-FIX} \quad f(fix(f)) \sqsubseteq fix(f) \]
\[ \text{fix}(f) \sqsubseteq f(fix(f)) \]
\[ f(fix(f)) = fix(f) \]
PROOFS WITH LEAST FIXED POINTS

**LFP-FIX**

\[ f(\text{fix}(f)) \subseteq \text{fix}(f) \]

**LFP-LEAST**

\[ \text{fix}(f) \subseteq d \]

Application: least pre-fixed points of monotone functions are (least) fixed points.

**LFP-FIX**

\[ f(\text{fix}(f)) \subseteq \text{fix}(f) \]

**LFP-LEAST**

\[ f(d) \subseteq d \]

**ASYM**

\[ f(\text{fix}(f)) \subseteq \text{fix}(f) \]

\[ \text{fix}(f) \subseteq f(\text{fix}(f)) \]

\[ f(\text{fix}(f)) = \text{fix}(f) \]

**MON**

\[ f(\text{fix}(f)) \subseteq \text{fix}(f) \]

\[ f(f(\text{fix}(f))) \subseteq f(\text{fix}(f)) \]

\[ \text{fix}(f) \subseteq f(\text{fix}(f)) \]