DENOTATIONAL SEMANTICS

Meven LENNON-BERTRAND
Lectures for Part II CST 2023/2024
• 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
WHAT IS THIS COURSE ABOUT?

• **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**
· **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 $\leftrightarrow$ Partial recursive function
Boolean circuit $\leftrightarrow$ Boolean function
...

Type $\mapsto$ Domain
Program $\mapsto$ Continuous functions between domains
Syntax $\rightarrow$ Semantics

Program $P$ $\leftrightarrow$ Denotation $[P]$

Recursive program $\leftrightarrow$ Partial recursive function

Boolean circuit $\leftrightarrow$ Boolean function

... $\rightarrow$

Type $\leftrightarrow$ Domain

Program $\leftrightarrow$ Continuous functions between domains
### 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.
- \( J_P \) represents the contribution of \( P \) to any program containing \( P \).
- Much more flexible than whole-program semantics.
### 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.
- $\llbracket P \rrbracket$ 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

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

Boolean expressions

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

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 \( \mathbb{L} \) 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

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

Commands

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

Arithmetic expressions

$$A \in Aexp ::= n \mid L \mid A + A \mid ...$$

Boolean expressions

$$B \in Bexp ::= true \mid false \mid A = A \mid \neg B \mid ...$$

Commands

$$C \in Comm ::= skip \mid L := A \mid C;C \mid if B then C else C \mid while B do C$$
Denotation functions — naïvely

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

where

\[ \mathbb{Z} = \{ ... , -1, 0, 1, ... \} \]
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?**

\[ A[n] = n \]
\[ A[L] = ??? \]
Denotation functions

State = (𝕃 → ℤ)

A : 𝐀𝐞𝐱𝐩 → (State → ℤ)

B : 𝐁𝐞𝐱𝐩 → (State → 𝔹)

C : 𝐂𝐨𝐦𝐦 → (State ⇀ State)

where ⇀ denotes partial functions and ℤ = {…, −1, 0, 1, …}

𝔹 = {true, false}. 
Denotation functions

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

\[ \mathcal{A} : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \]
\[ \mathcal{B} : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathbb{B}) \]

where

\[ \mathbb{Z} = \{ ..., -1, 0, 1, ... \} \]
\[ \mathbb{B} = \{ \text{true}, \text{false} \}. \]
Denotation functions

\begin{align*}
\text{State} &= (\mathbb{L} \to \mathbb{Z}) \\
\mathcal{A} : \text{Aexp} &\to (\text{State} \to \mathbb{Z}) \\
\mathcal{B} : \text{Bexp} &\to (\text{State} \to \mathbb{B}) \\
\mathcal{C} : \text{Comm} &\to (\text{State} \to \text{State})
\end{align*}

where $\rightarrow$ denotes partial functions and

\begin{align*}
\mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\
\mathbb{B} &= \{\text{true, false}\}.
\end{align*}
\begin{align*}
\mathcal{A}[n] &= \lambda s \in \text{State}. \ n \\
\mathcal{A}[A_1 + A_2] &= \lambda s \in \text{State}. \ \mathcal{A}[A_1](s) + \mathcal{A}[A_2](s)
\end{align*}
The semantic rules for arithmetic expressions are as follows:

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

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

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

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

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

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

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

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

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

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

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

This is compositionality!
$C[\text{skip}] = \lambda s \in \text{State. } s$

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

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 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...

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

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

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

\[
[\text{while } B \text{ do } C] = [\text{if } B \text{ then } (C;\text{while } B \text{ do } C) \text{ else skip}]
\]

\[
= \lambda s \in \text{State}. \text{if}(\llbracket B \rrbracket, \llbracket \text{while } B \text{ do } C \rrbracket \circ \llbracket C \rrbracket (s), s)
\]

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

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

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

\[
w \mapsto \lambda s \in \text{State}. \text{if}(b(s), w \circ c(s), s).
\]
• Why/when does $w = F_{b,c}(w)$ have a solution?
• What if it has several solutions? Which one should be our $\textbf{while } B \textbf{ do } C$?
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 \([\text{while } B \text{ do } C]\)?

Our occupation for the next few lectures...
INTRODUCTION

A TASTE OF DOMAIN THEORY
AN EXAMPLE

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

should be some \( w \) such that:

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

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

should be some \( w \) such that:

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

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

\[
F(w)([X \mapsto x, Y \mapsto y]) = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
w([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0.
\end{cases}
\]
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 $w'$. |
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 \( w(s) = w'(s) \).
- \( w \subseteq w' \) if the graph of \( w \) is included in the graph of \( w' \).

Least element \( \bot \in D \):

- \( \bot \) = totally undefined partial function
- \( \bot \) = partial function with empty graph
Define $w_n = F^n(w)$, that is

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


Approximating the fixed point

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

$$
\begin{align*}
w_0 &= \bot \\
\forall n \geq 0 : \quad w_{n+1} &= F(w_n).
\end{align*}
$$

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

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

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

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

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

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

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

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

$$
\begin{align*}
    w_0 &\sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots
\end{align*}
$$
Define \( w_n = F^n(w) \), that is
\[
\begin{align*}
  w_0 &= \perp \\
  w_{n+1} &= F(w_n).
\end{align*}
\]

\( 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} \]

\( w_0 \subseteq w_1 \subseteq \ldots \subseteq w_n \subseteq \ldots \subseteq w_\infty \)?
Define $w_n = F^n(w)$, that is

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

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

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

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

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

(by definition of \( F \))

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

(by definition of \( \omega_\infty \))

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

(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 \(\omega_\infty\))

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

\(\omega_\infty\) is a fixed point

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 $\subseteq$ that is

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

- **reflexive**: $\forall d \in D. \; d \sqsubseteq d$
- **transitive**: $\forall d, d', d'' \in D. \; d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$
- **antisymmetric**: $\forall d, d' \in D. \; d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.
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 $Y$;
Underlying set: partial functions $f$ with domain of definition $\text{dom}(f) \subseteq X$ and taking values in $Y$;

Order: $f \preceq g$ 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 $f: D \rightarrow E$ between posets is monotone if

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

$$\forall d, d' \in D. \quad d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$

$$\text{MON} \quad \frac{x \sqsubseteq y}{f(x) \sqsubseteq f(y)}$$
LEAST FIXED POINTS

LEAST ELEMENTS AND PRE-FIXED POINTS
An element $d \in S$ is the \textbf{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 \( \bot \).
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 \( \bot \).

\[
\begin{align*}
\text{LEAST} & \quad x \in S \\
\text{ASYM} & \quad \bot_S \sqsubseteq x \\
\text{LEAST} & \quad \bot_S \sqsubseteq \bot'_S \\
\text{LEAST} & \quad \bot_S \sqsubseteq \bot_S \\
\text{LEAST} & \quad \bot'_S \sqsubseteq \bot_S \\
\end{align*}
\]

\( \bot_S = \bot'_S \)
An element \( d \in D \) is a pre-fixed point of \( f \) if it satisfies \( f(d) \sqsubseteq d \).
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

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

\[
\begin{align*}
\text{LFP-FIX} & \quad f(\text{fix}(f)) \sqsubseteq \text{fix}(f) \\
\text{LFP-LEAST} & \quad f(d) \sqsubseteq d \implies \text{fix}(f) \sqsubseteq d
\end{align*}
\]
**Proofs with least fixed points**

\[ \text{LFP-FIX} \quad f(\text{fix}(f)) \subseteq \text{fix}(f) \]

The least pre-fixed point is a fixed point.
To prove $\text{fix}(f) \subseteq d$, it is enough to show $f(d) \subseteq d$. 

**Proofs with least fixed points**

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

**LFP-LEAST**

\[
\frac{f(d) \subseteq d}{\text{fix}(f) \subseteq d}
\]
PROOFS WITH LEAST FIXED POINTS

LFP-FIX
\[ f(\text{fix}(f)) \subseteq \text{fix}(f) \]

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

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

\[
\begin{align*}
\text{LFP-FIX} & : f(\text{fix}(f)) \subseteq \text{fix}(f) \\
\text{ASYM} & : f(\text{fix}(f)) \subseteq \text{fix}(f) \\
\text{LFP-FIX} & : f(\text{fix}(f)) \subseteq \text{fix}(f) \\
\text{fix}(f) & \subseteq f(\text{fix}(f)) \\
& \Rightarrow f(\text{fix}(f)) = \text{fix}(f)
\end{align*}
\]
PROOFS WITH LEAST FIXED POINTS

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

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

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

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

MON
\[ f(f(\text{fix}(f))) \sqsubseteq f(\text{fix}(f)) \]

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

\[ f(\text{fix}(f)) = \text{fix}(f) \]
The least upper bound of countable increasing chains $d_0 \subseteq d_1 \subseteq d_2 \subseteq \ldots$, written $\bigsqcup_{n \geq 0} d_n$, satisfies the two following properties:

**LUB-BOUND**

$$x_i \subseteq \bigsqcup_{n \geq 0} x_n$$

**LUB-LEAST**

$$\forall n \geq 0. x_n \subseteq x$$
Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}$, $d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$.

For any $d$, $\bigsqcup_n d = d$.

For any chain and $N \in \mathbb{N}$, $\bigsqcup_n d_n = \bigsqcup_{n+N} d_n$. 
Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}$. $d_n \subseteq e_n$, then $\bigsqcup_n d_n \subseteq \bigsqcup_n e_n$. 
Lubs are unique.

Lubs are monotone: if for all \( n \in \mathbb{N} \), \( d_n \subseteq e_n \), then \( \bigsqcup_n d_n \subseteq \bigsqcup_n e_n \).

\[
\begin{array}{c}
\forall i. \ d_i \subseteq e_i \\
\hline
\hline
\hline
\end{array}
\]

LUB-MON

\[ \bigsqcup_n d_n \subseteq \bigsqcup_n e_n \]
Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}$. $d_n \subseteq e_n$, then $\bigsqcup_n d_n \subseteq \bigsqcup_n e_n$.

For any $d$, $\bigsqcup_n d = d$. 
PROPERTIES OF LUBS

Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}$. $d_n \subseteq e_n$, then $\bigsqcup_n d_n \subseteq \bigsqcup_n e_n$.

For any $d$, $\bigsqcup_n d = d$.

For any chain and $N \in \mathbb{N}$, $\bigsqcup_n d_n = \bigsqcup_n d_{n+N}$. 
Lubs are unique (if they exist).

Lubs are monotone: if for all \( n \in \mathbb{N} \). \( d_n \subseteq e_n \), then \( \bigsqcup_n d_n \subseteq \bigsqcup_n e_n \) (if they exist).

For any \( d \), \( \bigsqcup_n d = d \) (and in particular it exists).

For any chain and \( N \in \mathbb{N} \), \( \bigsqcup_n d_n = \bigsqcup_n d_{n+N} \) (if any of the two exists).
Assume $d_{m,n} \in D \ (m, n \geq 0)$ satisfies

$$m \leq m' \land n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'}.$$
Assume $d_{m,n} \in D \ (m, n \geq 0)$ satisfies
\[ m \leq m' \land n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'} . \] (†)

Then, assuming they exist, the lubs form two chains
\[
\bigsqcup_{n \geq 0} d_{0,n} \sqsubseteq \bigsqcup_{n \geq 0} d_{1,n} \sqsubseteq \bigsqcup_{n \geq 0} d_{2,n} \sqsubseteq \ldots
\]
and
\[
\bigsqcup_{m \geq 0} d_{m,0} \sqsubseteq \bigsqcup_{m \geq 0} d_{m,1} \sqsubseteq \bigsqcup_{m \geq 0} d_{m,2} \sqsubseteq \ldots
\]
Assume $d_{m,n} \in D \ (m, n \geq 0)$ satisfies

$$m \leq m' \wedge n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'}.$$  \hfill (†)

Then, assuming they exist, the lubs form two chains

$$\bigsqcup_{n \geq 0} d_{0,n} \sqsubseteq \bigsqcup_{n \geq 0} d_{1,n} \sqsubseteq \bigsqcup_{n \geq 0} d_{2,n} \sqsubseteq \ldots$$

and

$$\bigsqcup_{m \geq 0} d_{m,0} \sqsubseteq \bigsqcup_{m \geq 0} d_{m,1} \sqsubseteq \bigsqcup_{m \geq 0} d_{m,2} \sqsubseteq \ldots$$

Moreover, again assuming they exist,

$$\bigsqcup_{m \geq 0} \left( \bigsqcup_{n \geq 0} d_{m,n} \right) = \bigsqcup_{k \geq 0} d_{k,k} = \bigsqcup_{n \geq 0} \left( \bigsqcup_{m \geq 0} d_{m,n} \right).$$
LEAST FIXED POINTS
COMPLETE PARTIAL ORDERS AND DOMAINS
A **chain complete poset/cpo** is a poset \((D, \sqsubseteq)\) in which all chains have least upper bounds.
A **chain complete poset/cpo** is a poset \((D, \sqsubseteq)\) in which all chains have least upper bounds.

**Beware**: the lub need only exist if the \(x_i\) form a chain!
A **chain complete poset/cpo** is a poset \((D, \sqsubseteq)\) in which all chains have least upper bounds.

Beware: the lub need only exist if the \(x_i\) form a chain!

A **domain** is a cpo with a least element \(\bot\).
Least element: \( \bot \) is the totally undefined function.
**Least element:** $\bot$ is the totally undefined function.

**Lub of a chain:** $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ has lub $f$ such that

$$f(x) = \begin{cases} f_n(x) & \text{if } x \in \text{dom}(f_n) \text{ for some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$
**Least element:** $\bot$ is the totally undefined function.

**Lub of a chain:** $f_0 \subseteq f_1 \subseteq f_2 \subseteq \ldots$ has lub $f$ such that

$$f(x) = \begin{cases} f_n(x) & \text{if } x \in \text{dom}(f_n) \text{ for some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

**Beware:** the definition of $\bigsqcup_{n \geq 0} f_n$ is unambiguous only if the $f_i$ form a chain!
The flat natural numbers $\mathbb{N}_\bot$
LEAST FIXED POINTS
CONTINUOUS FUNCTIONS
Given two cpos $D$ and $E$, a function $f: D \to E$ is **continuous** if

- it is monotone, and
- it preserves lubs of chains, i.e. for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ in $D$, we have

$$f(\bigsqcup_{n\geq 0} d_n) = \bigsqcup_{n\geq 0} f(d_n)$$
Continuity and strictness

Given two cpos $D$ and $E$, a function $f: D \rightarrow E$ is **continuous** if

- it is monotone, and
- it preserves lubs of chains, i.e. for all chains $d_0 \sqsubseteq d_1 \sqsubseteq ...$ in $D$, we have

\[
f(\bigsqcup_{n \geq 0} d_n) = \bigsqcup_{n \geq 0} f(d_n)
\]

A function $f$ is **strict** if $f(\bot_D) = \bot_E$. 
All computable functions are continuous.
All *computable* functions are continuous.
All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”?

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 ⊥ ...</td>
<td>⊥</td>
</tr>
<tr>
<td>0 0 0 0 1 ...</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>
All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”? 

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 ⊥ ...</td>
<td>⊥</td>
</tr>
<tr>
<td>0 0 0 0 1 ...</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 ...</td>
<td>?</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>

Intuition: non-continuity ≈ “jump at infinity” ≈ non-computability

Later in the course: show the thesis… by giving a denotational semantics.
All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”?

```
0 0 ⊥ ...  ↦ ⊥
0 0 0 0 1 ...  ↦ 1
0 0 0 0 0 0 0 0 ⊥ ...  ↦ ⊥
0 0 0 0 0 0 0 0 0 ...  ↦ ?
0 0 0 0 0 0 0 0 0 0 ...  ↦ 0
```
All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”?

\[
\begin{array}{ccccccc}
0 & 0 & \bot & \ldots & & & \\
0 & 0 & 0 & 0 & 1 & \ldots & \mapsto 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mapsto \bot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & \mapsto ? \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mapsto 0 \\
\end{array}
\]

Intuition: non-continuity \(\approx\) “jump at infinity” \(\approx\) non-computability
All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”?

\[
\begin{array}{cccccccc}
0 & 0 & \bot & \ldots & \mapsto & \bot \\
0 & 0 & 0 & 0 & 1 & \ldots & \mapsto & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \bot & \ldots & \mapsto & \bot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & \mapsto & ? \\
0 & 0 & 0 & 0 & 0 & 0 & \bar{0} & \mapsto & 0 \\
\end{array}
\]

Intuition: non-continuity \(\approx\) “jump at infinity” \(\approx\) non-computability

 Later in the course: **show** the thesis... by giving a denotational semantics.
LEAST FIXED POINTS
KLEENE’S FIXED POINT THEOREM
Kleene’s fixed point theorem

Let \( f: D \to D \) be a continuous function on a domain \( D \). Then \( f \) possesses a least pre-fixed point, given by

\[
\text{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot).
\]
Kleene’s fixed point theorem

Let $f: D \rightarrow D$ be a continuous function on a domain $D$. Then $f$ possesses a least pre-fixed point, given by

$$\text{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot).$$

It is thus also the least fixed point of $f$!
CONSTRUCTIONS ON DOMAINS
CONSTRUCTIONS ON DOMAINS

FLAT DOMAINS
The flat domain on a set $X$ is defined by:

- its underlying set $X \cup \{\bot\}$;
- $x \sqsubseteq x'$ if either $x = \bot$ or $x = x'$.
Let $f : X \rightarrow Y$ be a partial function between two sets. Then

$$f_\perp : X_\perp \rightarrow Y_\perp$$

$$d \mapsto \begin{cases} f(d) & \text{if } d \in X \text{ and } f \text{ is defined at } d \\ \perp & \text{if } d \in X \text{ and } f \text{ is not defined at } d \\ \perp & \text{if } d = \perp \end{cases}$$

defines a continuous function between the corresponding flat domains.
CONSTRUCTIONS ON DOMAINS

PRODUCTS OF DOMAINS
The **product** of two posets \((D_1, \sqsubseteq_1)\) and \((D_2, \sqsubseteq_2)\) has underlying set

\[
D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \land d_2 \in D_2\}
\]

and partial order \(\sqsubseteq\) defined by

\[
(d_1, d_2) \sqsubseteq (d_1', d_2') \iff d_1 \sqsubseteq_1 d_1' \land d_2 \sqsubseteq_2 d_2'
\]
The **product** of two posets \((D_1, \sqsubseteq_1)\) and \((D_2, \sqsubseteq_2)\) has underlying set

\[ D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \land d_2 \in D_2\} \]

and partial order \(\sqsubseteq\) defined by

\[(d_1, d_2) \sqsubseteq (d_1', d_2') \overset{\text{def}}{\iff} d_1 \sqsubseteq_1 d_1' \land d_2 \sqsubseteq_2 d_2' \]
lubs of chains are computed componentwise:

\[
\bigsqcup_{n \geq 0} (d_{1,n}, d_{2,n}) = (\bigsqcup_{i \geq 0} d_{1,i}, \bigsqcup_{j \geq 0} d_{2,j}).
\]
COMPONENTWISE LUBS AND LEAST ELEMENTS

Lubs of chains are computed componentwise:

\[
\bigvee_{n \geq 0} (d_{1,n}, d_{2,n}) = \left( \bigvee_{i \geq 0} d_{1,i}, \bigvee_{j \geq 0} d_{2,j} \right).
\]

If \((D_1, \sqsubseteq_1)\) and \((D_2, \sqsubseteq_2)\) have least elements, so does \((D_1 \times D_2, \sqsubseteq)\) with

\[
\bot_{D_1 \times D_2} = (\bot_{D_1}, \bot_{D_2}).
\]
COMPONENTWISE LUBS AND LEAST ELEMENTS

Lubs of chains are computed componentwise:

\[
\bigsqcup_{n \geq 0} (d_{1,n}, d_{2,n}) = (\bigsqcup_{i \geq 0} d_{1,i}, \bigsqcup_{j \geq 0} d_{2,j}).
\]

If \((D_1, \sqsubseteq_1)\) and \((D_2, \sqsubseteq_2)\) have least elements, so does \((D_1 \times D_2, \sqsubseteq)\) with

\[
\perp_{D_1 \times D_2} = (\perp_{D_1}, \perp_{D_2})
\]

Products of cpos (domains) are cpos (domains).
A function $f : (D \times E) \rightarrow F$ is monotone if and only if it is monotone in each argument separately:

\[
\forall d, d' \in D, e \in E. d \sqsubseteq d' \Rightarrow f(d, e) \sqsubseteq f(d', e)
\]

\[
\forall d \in D, e, e' \in E. e \sqsubseteq e' \Rightarrow f(d, e) \sqsubseteq f(d, e').
\]
A function $f : (D \times E) \to F$ is monotone if and only if it is monotone in each argument separately:

$$\forall d, d' \in D, e \in E. d \sqsubseteq d' \Rightarrow f(d, e) \sqsubseteq f(d', e)$$
$$\forall d \in D, e, e' \in E. e \sqsubseteq e' \Rightarrow f(d, e) \sqsubseteq f(d, e').$$

Moreover, it is continuous if and only if it preserves lubs in each argument separately:

$$f(\bigsqcup_{m \geq 0} d_m, e) = \bigsqcup_{m \geq 0} f(d_m, e)$$
$$f(d, \bigsqcup_{n \geq 0} e_n) = \bigsqcup_{n \geq 0} f(d, e_n).$$
DERIVED RULES FOR FUNCTIONS OF TWO ARGUMENTS

\[
\text{MON}\times f \text{ monotone } x \sqsubseteq x' \quad y \sqsubseteq y' \\
f(x, y) \sqsubseteq f(x', y')
\]

\[
f\left(\bigsqcup_m x_m, \bigsqcup_n y_n\right) = \bigsqcup_m \bigsqcup_n f(x_m, y_n) = \bigsqcup_k f(x_k, y_k)
\]
Let $D_1$ and $D_2$ be cpos. The projections

\[ \pi_1 : D_1 \times D_2 \rightarrow D_1 \quad (d_1, d_2) \mapsto d_1 \]

\[ \pi_2 : D_1 \times D_2 \rightarrow D_2 \quad (d_1, d_2) \mapsto d_2 \]

are continuous functions.
Let $D_1$ and $D_2$ be cpos. The projections

$$
\pi_1 : \quad D_1 \times D_2 \rightarrow D_1 \quad (d_1, d_2) \mapsto d_1
$$

$$
\pi_2 : \quad D_1 \times D_2 \rightarrow D_2 \quad (d_1, d_2) \mapsto d_2
$$

are continuous functions.

If $f_1 : D \rightarrow D_1$ and $f_2 : D \rightarrow D_2$ are continuous functions from a cpo $D$, then the pairing function

$$
\langle f_1, f_2 \rangle : \quad D \rightarrow \quad D_1 \times D_2 \quad d \mapsto (f_1(d), f_2(d))
$$

is continuous.
The conditional function

\[
\text{if} : \mathbb{B}_\bot \times (D \times D) \rightarrow D \\
(x, d) \mapsto \begin{cases} 
\pi_1(d) & \text{if } x = \text{true} \\
\pi_2(d) & \text{if } x = \text{false} \\
\bot_D & \text{if } x = \bot
\end{cases}
\]

is continuous.
Given a set $I$, suppose that for each $i \in I$ we are given a set $X_i$. The (cartesian) product of the $X_i$ is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\ldots, x_i, \ldots)_{i \in I}$ such that $x_i \in X_i$;

Special case: $\prod_{i \in \mathbb{B}} D_i$ corresponds to $D_{\text{true}} \times D_{\text{false}}$. 

Projections (for any $i \in I$):

$$\pi_i: \left( \prod_{i \in I} X_i \right) \to X_i$$
Given a set $I$, suppose that for each $i \in I$ we are given a set $X_i$. The (cartesian) product of the $X_i$ is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\ldots, x_i, \ldots)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: $p$ defined on $I$ such that $p(i) \in X_i$. 

Given a set $I$, suppose that for each $i \in I$ we are given a set $X_i$. The (cartesian) product of the $X_i$ is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\ldots, x_i, \ldots)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: $p$ defined on $I$ such that $p(i) \in X_i$.

Special case: $\prod_{i \in \mathbb{B}} D_i$ corresponds to $D_{\text{true}} \times D_{\text{false}}$. 

Given a set $I$, suppose that for each $i \in I$ we are given a set $X_i$. The (cartesian) product of the $X_i$ is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(..., x_i, ...)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: $p$ defined on $I$ such that $p(i) \in X_i$.

Special case: $\prod_{i \in \mathbb{B}} D_i$ corresponds to $D_{\text{true}} \times D_{\text{false}}$.

Projections (for any $i \in I$):

$$\pi_i : \left( \prod_{i \in I} X_i \right) \to X_i$$
Given a set $I$, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The product of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;
Given a set $I$, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The **product** of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;
- componentwise order

\[ p \sqsubseteq p' \overset{\text{def}}{\iff} \forall i \in I. \ p_i \sqsubseteq_i p'_i. \]
Given a set $I$, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The **product** of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;
- componentwise order

\[
p \sqsubseteq p' \iff \forall i \in I. p_i \sqsubseteq_i p'_i.
\]

$I$-indexed products of cpos (domains) are cpos (domains), and projections are continuous.
CONSTRUCTIONS ON DOMAINS

FUNCTION DOMAINS
Given two cpos $\langle D, \sqsubseteq_D \rangle$ and $\langle E, \sqsubseteq_E \rangle$, the function $\text{cpo} \ (D \to E, \sqsubseteq)$ has underlying set

$$\{ f : D \to E \mid \text{is a continuous function} \}$$

equipped with the pointwise order:

$$f \sqsubseteq f' \iff \forall d \in D. \ f(d) \sqsubseteq_E f'(d).$$
Given two cpos \((D, \sqsubseteq_D)\) and \((E, \sqsubseteq_E)\), the function cpo \((D \to E, \sqsubseteq)\) has underlying set
\[
\{ f : D \to E \mid \text{is a continuous function} \}
\]
equipped with the pointwise order:
\[
f \sqsubseteq f' \iff \forall d \in D. \ f(d) \sqsubseteq_E f'(d).
\]

\[
\begin{align*}
f \sqsubseteq_{D \to E} g & \quad x \sqsubseteq_D y \\
\hline
f(x) \sqsubseteq_E g(y)
\end{align*}
\]
Given two cpos \((D, \sqsubseteq_D)\) and \((E, \sqsubseteq_E)\), the function \(\text{cpo } (D \to E, \sqsubseteq)\) has underlying set

\[
\{ f : D \to E \mid \text{is a continuous function} \}
\]
equipped with the pointwise order:

\[
f \sqsubseteq f' \iff \forall d \in D. \ f(d) \sqsubseteq_E f'(d).
\]

Argumentwise least elements and lubs:

\[
\bot_{D \to E}(d) = \bot_E \quad \quad \left( \bigsqcup_{n \geq 0} f_n \right)(d) = \bigsqcup_{n \geq 0} f_n(d)
\]
Evaluation, currying \((f : (D' \times D) \to E)\) and composition

\[
\text{eval} : (D \to E) \times D \to E \\
(f, d) \mapsto f(d)
\]

\[
\text{cur}(f) : D' \to (D \to E) \\
d' \mapsto \lambda d \in D. f(d', d)
\]

\[
\circ : ((E \to F) \times (D \to E)) \to (D \to F) \\
(f, g) \mapsto \lambda d \in D. g(f(d))
\]

are all well-defined and continuous.
contituency of the fixed point operator

\[ \text{fix}: (D \to D) \to D \]

is continuous.
CONSTRUCTIONS ON DOMAINS

BACK TO THE INTRODUCTION
\[\text{while } X > 0 \text{ do } (Y := X \times Y; X := X - 1)\]

is a fixed point of the following \( F : D \rightarrow D \), where \( D \) is \((\text{State} \rightarrow \text{State})\):

\[
F(w)([X \mapsto x, Y \mapsto y]) = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
w([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0.
\end{cases}
\]
[while $X > 0$ do ($Y := X \cdot Y; X := X - 1$)]

is a fixed point of the following $F : D \rightarrow D$, where $D$ is ($\text{State}_\bot \rightarrow \text{State}_\bot$):

$$F(w)([X \mapsto x, Y \mapsto y]) = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0. \end{cases}$$

$$F(\bot) = \bot$$

$\text{State}_\bot \rightarrow \text{State}_\bot$ is a domain!
Kleene’s fixed point theorem:

\[ w_\infty = \bigwedge_{i \in \mathbb{N}} F^n(\bot) \]

is the least fixed point of \( F \), and in particular a fixed point.
Kleene's fixed point theorem:

\[ w_\infty = \bigsqcup_{i \in \mathbb{N}} F^n(\bot) \]

is the least fixed point of \( F \), and in particular a fixed point.

We can compute explicitly

\[ w_\infty[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } x \geq 0 \end{cases} \]

And check this agrees with the operational semantics.
Let $D$ be a domain, $f: D \rightarrow D$ be a continuous function and $S \subseteq D$ be a subset of $D$. If the set $S$

(i) contains $\bot$,

(ii) is stable under $f$, i.e. $f(S) \subseteq S$,

(iii) is chain-closed, i.e. the lub of any chain of elements of $S$ is also in $S$,

then $\text{fix}(f) \in S$. 

\[ \Phi(\bot) \Rightarrow \Phi(x) \Rightarrow \Phi(f(x)) \]
\[ (\forall i \in \mathbb{N}. \Phi(x_i) \Rightarrow \Phi(\bigsqcup_{i \in \mathbb{N}} x_i)) \Rightarrow \Phi(\text{fix}(f)) \]
Reasoning on fixed points: Scott induction

Let $D$ be a domain, $f: D \rightarrow D$ be a continuous function and $S \subseteq D$ be a subset of $D$. If the set $S$

(i) contains $\bot$,
(ii) is stable under $f$, i.e. $f(S) \subseteq S$,
(iii) is chain-closed, i.e. the lub of any chain of elements of $S$ is also in $S$,

then $\text{fix}(f) \in S$.

\[
\Phi(\bot) \quad \Phi(x) \Rightarrow \Phi(f(x)) \quad (\forall i \in \mathbb{N}. \Phi(x_i)) \Rightarrow \Phi(\bigsqcup_{i \in \mathbb{N}} x_i)
\]

ScottInd

\[
\Phi(\text{fix}(f))
\]
All the following are chain-closed:

\[
\{(x, y) \in D \times D \mid x \preceq y\}, \quad d \downarrow \defeq \{x \in D \mid x \preceq d\}
\]

and

\[
\{(x, y) \in D \times D \mid x = y\}
\]

\(f^{-1}(S) = \{x \in D \mid f(x) \in S\}\) if \(S \subseteq E\) is chain-closed, and \(f : D \to E\) is continuous.

\(S \cup T\) and \(\bigcap_{i \in I} S_i\) if \(S, T\) and \(S_i\) are

\[
\forall S \defeq \{y \in E \mid \forall x \in D. (x, y) \in S\} \subseteq E
\]

if \(S \subseteq D \times E\) is
All the following are chain-closed:

\[ \{(x, y) \in D \times D \mid x \sqsubseteq y\}, \quad d \downarrow \overset{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\} \quad \text{and} \quad \{(x, y) \in D \times D \mid x = y\} \]
All the following are chain-closed:

\[
\{(x, y) \in D \times D \mid x \sqsubseteq y\}, \quad d \overset{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\} \quad \text{and} \quad \{(x, y) \in D \times D \mid x = y\}
\]

\[
f^{-1}S = \{x \in D \mid f(x) \in S\} \quad \text{if } S \subseteq E \text{ is chain-closed, and } f : D \to E \text{ is continuous}
\]
All the following are chain-closed:

\[(x, y) \in D \times D \mid x \sqsubseteq y\] , \[d \overset{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\}\] and \[\{(x, y) \in D \times D \mid x = y\}\]

\[f^{-1}S = \{x \in D \mid f(x) \in S\}\] if \(S \subseteq E\) is chain-closed, and \(f: D \to E\) is continuous

\[S \cup T\] and \[\bigcap_{i \in I} S_i\] if \(S, T\) and \(S_i\) are
All the following are chain-closed:

\{(x, y) \in D \times D \mid x \sqsubseteq y\} , \quad d \overset{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\} \quad \text{and} \quad \{(x, y) \in D \times D \mid x = y\}

\[ f^{-1}S = \{x \in D \mid f(x) \in S\} \quad \text{if} \ S \subseteq E \text{ is chain-closed, and} \ f : D \to E \text{ is continuous} \]

\[ S \cup T \quad \text{and} \quad \bigcap_{i \in I} S_i \quad \text{if} \ S, T \text{ and } S_i \text{ are} \]

\[ \forall S \overset{\text{def}}{=} \{y \in E \mid \forall x \in D. (x, y) \in S\} \subseteq E \quad \text{if} \ S \subseteq D \times E \text{ is} \]
Assume \( f(d) \sqsubseteq d \), i.e. \( d \) is a pre-fixed point of the continuous \( f : D \to D \). By Scott induction on \( d \downarrow \), \( \text{fix}(f) \sqsubseteq d \).
Assume $f(d) \sqsubseteq d$, i.e. $d$ is a pre-fixed point of the continuous $f : D \rightarrow D$. By Scott induction on $d \downarrow$, $\text{fix}(f) \sqsubseteq d$.

**Proof!**
Let $w_\infty : \text{State}_\perp \to \text{State}_\perp$ be the denotation of

$$\text{while } X > 0 \text{ do } (Y := X \times Y; X := X - 1)$$

Recall that $w_\infty = \text{fix}(F)$ where

$$F(w)(x, y) = \begin{cases} (x, y) & \text{if } x \leq 0 \\ w(x - 1, x \cdot y) & \text{if } x > 0 \end{cases}$$

$$F(w)(\perp) = \perp$$
Let $\omega_\infty : \text{State}_\bot \rightarrow \text{State}_\bot$ be the denotation of

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

Recall that $\omega_\infty = \text{fix}(F)$ where

$$F(w)(x, y) = \begin{cases} (x, y) & \text{if } x \leq 0 \\ w(x - 1, x \cdot y) & \text{if } x > 0 \end{cases}$$

$F(w)(\bot) = \bot$

Claim:

$$\forall x. \forall y \geq 0. \omega_\infty(x, y) \Downarrow \implies \pi_Y(\omega_\infty(x, y)) \geq 0$$
Let $w_\infty : \text{State}_\bot \rightarrow \text{State}_\bot$ be the denotation of

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

Recall that $w_\infty = \text{fix}(F)$ where

$$F(w)(x, y) = \begin{cases} (x, y) & \text{if } x \leq 0 \\ w(x - 1, x \cdot y) & \text{if } x > 0 \end{cases}$$

$$F(w)(\bot) = \bot$$

Claim:

$$\forall x. \forall y \geq 0. w_\infty (x, y) \Downarrow \implies \pi_Y (w_\infty (x, y)) \geq 0$$

Proof: by Scott induction!
PCF
PCF

Terms and Types
Types: \[ \tau ::= \text{nat} | \text{bool} | \tau \rightarrow \tau \]
Types: \( \tau ::= \text{nat} \mid \text{bool} \mid \tau \rightarrow \tau \)

Terms: \( t ::= 0 \mid \text{succ}(t) \mid \text{pred}(t) \mid \text{true} \mid \text{false} \mid \text{zero?}(t) \mid \text{if } t \text{ then } t \text{ else } t \mid x \mid \text{fun } x : \tau . t \mid t \; t \mid \text{fix}(t) \)
The term $t$ has type $\tau$ in context $\Gamma$.

\[
\begin{align*}
\text{ZERO} & : \quad \frac{}{\Gamma \vdash 0 : \text{nat}} \\
\text{Succ} & : \quad \frac{\Gamma \vdash t : \text{nat}}{\Gamma \vdash \text{succ}(t) : \text{nat}} \\
\text{Pred} & : \quad \frac{\Gamma \vdash t : \text{nat}}{\Gamma \vdash \text{pred}(t) : \text{nat}}
\end{align*}
\]
The term $t$ has type $\tau$ in context $\Gamma$.

- **ZERO**
  \[
  \Gamma \vdash 0 : \text{nat}
  \]

- **SUCC**
  \[
  \Gamma \vdash t : \text{nat} \\
  \Gamma \vdash \text{succ}(t) : \text{nat}
  \]

- **PRED**
  \[
  \Gamma \vdash t : \text{nat} \\
  \Gamma \vdash \text{pred}(t) : \text{nat}
  \]

- **TRUE**
  \[
  \Gamma \vdash \text{true} : \text{bool}
  \]

- **FALSE**
  \[
  \Gamma \vdash \text{false} : \text{bool}
  \]

- **ISZ**
  \[
  \Gamma \vdash t : \text{nat} \\
  \Gamma \vdash \text{zero}?(t) : \text{bool}
  \]

- **IF**
  \[
  \Gamma \vdash b : \text{bool} \\
  \Gamma \vdash t : \tau \\
  \Gamma \vdash t' : \tau \\
  \Gamma \vdash \text{if } b \text{ then } t \text{ else } t' : \tau
  \]
Typing for PCF (II)

\[ \Gamma(x) = \tau \]

**VAR**

\[ \Gamma \vdash x : \tau \]

**FUN**

\[ \Gamma, x:\sigma \vdash t : \tau \]

\[ \Gamma \vdash \text{fun } x:\sigma. t : \sigma \rightarrow \tau \]

**APP**

\[ \Gamma \vdash f : \sigma \rightarrow \tau \quad \Gamma \vdash u : \sigma \]

\[ \Gamma \vdash f \ u : \tau \]

**FIX**

\[ \Gamma \vdash f : \tau \rightarrow \tau \]

\[ \Gamma \vdash \text{fix}(f) : \tau \]
Typing for PCF (II)

\[
\begin{align*}
\text{VAR} & : \quad \Gamma(x) = \tau \\
\frac{}{\Gamma \vdash x : \tau} \\
\text{FUN} & : \quad \Gamma, x: \sigma \vdash t : \tau \\
\frac{}{\Gamma \vdash \text{fun} x: \sigma. t : \sigma \to \tau} \\
\text{APP} & : \quad \Gamma \vdash f : \sigma \to \tau \quad \Gamma \vdash u : \sigma \\
\frac{}{\Gamma \vdash f u : \tau} \\
\text{FIX} & : \quad \Gamma \vdash f : \tau \to \tau \\
\frac{}{\Gamma \vdash \text{fix}(f) : \tau}
\end{align*}
\]

\[
\begin{align*}
\text{PCF}_{\Gamma,\tau} & \overset{\text{def}}{=} \{ t \mid \Gamma \vdash t : \tau \} \\
\text{PCF}_\tau & \overset{\text{def}}{=} \text{PCF}_{\cdot,\tau}
\end{align*}
\]
Values:

\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun } x: \tau. t \]
Values:
\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun } x: \tau. \ t \]

\[ \text{VAL} \quad \frac{}{\text{\vdash } v : \tau} \]
\[ \frac{}{v \Downarrow^\tau v} \]
Values:

\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun } x: \tau . t \]

\[
\begin{align*}
\text{VAL} & \quad \vdash v : \tau \\
\Downarrow_\tau & \quad v \Downarrow_\tau v \\
\text{SUCC} & \quad t \Downarrow_{\text{nat}} v \\
& \quad \text{succ}(t) \Downarrow_{\text{nat}} \text{succ}(v) \\
\text{PRED} & \quad t \Downarrow_{\text{nat}} \text{succ}(v) \\
& \quad \text{pred}(t) \Downarrow_{\text{nat}} v
\end{align*}
\]
Values:

\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun } x: \tau. t \]

\[ \vdash v : \tau \]

\[ v \Downarrow^{\tau} v \]

\[ t \Downarrow^{\text{nat}} v \]

\[ \text{succ}(t) \Downarrow^{\text{nat}} \text{succ}(v) \]

\[ \text{pred}(t) \Downarrow^{\text{nat}} v \]

\[ t \Downarrow^{\text{nat}} 0 \]

\[ \text{zero?}(t) \Downarrow^{\text{bool}} \text{true} \]

\[ b \Downarrow^{\text{bool}} \text{true} \]

\[ \text{if } b \text{ then } t_1 \text{ else } t_2 \Downarrow^{\tau} v \]
PCF EVALUATION

Values:

\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun} \ x: \tau. \ t \]

\[
\begin{align*}
\text{VAL} & \quad \vdash v : \tau \\
\Rightarrow & \quad v \Downarrow^\tau v \\
\text{SUCC} & \quad t \Downarrow^\text{nat} v \\
\Rightarrow & \quad \text{succ}(t) \Downarrow^\text{nat} \text{succ}(v) \\
\text{PRED} & \quad t \Downarrow^\text{nat} \text{succ}(v) \\
\Rightarrow & \quad \text{pred}(t) \Downarrow^\text{nat} v \\
\text{ZEROZ} & \quad t \Downarrow^\text{nat} 0 \\
\Rightarrow & \quad \text{zero?}(t) \Downarrow^\text{bool} \text{true} \\
\text{FUN} & \quad t \Downarrow^\sigma \rightarrow^\tau \text{fun} \ x: \sigma. \ t' \\
\Rightarrow & \quad t \hspace{1mm} u \Downarrow^\tau v \\
\text{IFT} & \quad b \Downarrow^\text{bool} \text{true} \\
\Rightarrow & \quad \text{if} \ b \text{ then } t_1 \text{ else } t_2 \Downarrow^\tau v \\
\text{FIX} & \quad t \hspace{1mm} (\text{fix}(t)) \Downarrow^\tau v \\
\Rightarrow & \quad \text{fix}(t) \Downarrow^\tau v
\end{align*}
\]
Values:
\[ v ::= 0 \mid \text{succ}(v) \mid \text{true} \mid \text{false} \mid \text{fun} \ x: \tau. \ t \]

\[ \begin{align*}
\text{VAL} & : v : \tau \vdash v : \tau \\
\text{Succ} & : t \downarrow_{\text{nat}} v \quad \text{succ}(t) \downarrow_{\text{nat}} \text{succ}(v) \\
\text{Pred} & : t \downarrow_{\text{nat}} \text{succ}(v) \quad \text{pred}(t) \downarrow_{\text{nat}} v \\
\text{ZeroZ} & : t \downarrow_{\text{nat}} 0 \quad \text{zero?}(t) \downarrow_{\text{bool}} \text{true} \\
\text{Fun} & : t \downarrow_{\sigma \rightarrow \tau} \text{fun} \ x: \sigma. \ t' \quad t'[u/x] \downarrow_{\tau} v \\
\text{Fix} & : t(\text{fix}(t)) \downarrow_{\tau} v
\end{align*} \]

Alternatively: small-step \( t \leadsto_{\tau} u \), we have \( t \downarrow_{\tau} v \) iff \( t \leadsto^*_{\tau} u \).
plus  \text{def} \text{ = fun } x: \text{nat. fix(fun}(p: \text{nat }\rightarrow \text{nat})(y: \text{nat}). \\
\text{  if zero?(y) then } x \text{ else succ}(p \text{ pred(y)}))
\begin{align*}
\text{plus } 3 \ 1 & \Downarrow_{\text{nat}} 4
\end{align*}
plus \overset{\text{def}}{=} \text{fun } x : \text{nat}. \text{fix(fun}(p : \text{nat }\to \text{nat})(y : \text{nat}).
\text{if zero?}(y) \text{ then } x \text{ else } \text{succ}(p \text{ pred}(y)))

\text{plus } 3 \downarrow_\text{nat} 4

\Omega_\tau \overset{\text{def}}{=} \text{fix(fun } x : \tau. x)

\Omega_\tau \uparrow_\tau \quad \text{(diverges)}
    if zero?(y) then x else succ(p pred(y)))

plus 3 1 \downarrow_{\text{nat}} 4

Ωτ = fix(fun x: τ. x)

Ωτ \uparrow_τ \quad \text{(diverges)}

Try it out!
PCF is **Turing-complete**: for every partial recursive function \( \phi \), there is a PCF term \( \_ \) \( \phi \) such that for all \( n \in \mathbb{N} \), if \( \phi(n) \) is defined then \( \_ n \Downarrow_{\text{nat}} \phi(n) \).
PCF is Turing-complete: for every partial recursive function $\phi$, there is a PCF term $\overline{\phi}$ such that for all $n \in \mathbb{N}$, if $\phi(n)$ is defined then $\overline{\phi \ n} \downarrow_{\text{nat}} \phi(n)$.

(Later on: $\phi = \left[ \phi \right]$.)
Evaluation in PCF is **deterministic**: if both $t \Downarrow_{\tau} \nu$ and $t \Downarrow_{\tau} \nu'$ hold, then $\nu = \nu'$. 
Evaluation in PCF is **deterministic**: if both $t \Downarrow_\tau \nu$ and $t \Downarrow_\tau \nu'$ hold, then $\nu = \nu'$.

By (rule) induction on evaluation $\Downarrow$:

$$\{(t, \tau, \nu) \mid t \Downarrow_\tau \nu \land \forall \nu'. (t \Downarrow_\tau \nu' \implies \nu = \nu')\}$$

Intuition: there is always exactly one rule which applies.
PCF

CONTEXTUAL EQUIVALENCE
Two phrases of a programming language are contextually equivalent if any occurrences of the first phrase in a complete program can be replaced by the second phrase without affecting the observable results of executing the program.
Two phrases of a programming language are *contextually equivalent* if any occurrences of the first phrase in a *complete program* can be replaced by the second phrase without affecting the *observable results* of executing the program.

The intuitive notion of *program equivalence* for programmers.
\[ C ::= - \mid \text{succ}(C) \mid \text{pred}(C) \mid \text{zero}\?(C) \mid \begin{cases} \text{if } C \text{ then } t \text{ else } t \end{cases} \mid \begin{cases} \text{if } t \text{ then } C \text{ else } t \end{cases} \mid \begin{cases} \text{if } t \text{ then } t \text{ else } C \end{cases} \mid \text{fun } x : \tau . C \mid C t \mid t C \mid \text{fix}(C) \]
\[ C ::= \, - \, | \, \text{succ}(C) \, | \, \text{pred}(C) \, | \, \text{zero?}(C) \, | \, \text{if } C \text{ then } t \text{ else } t \, | \, \text{if } t \text{ then } C \text{ else } t \, | \, \text{if } t \text{ then } t \text{ else } C \, | \, \text{fun } x: \tau. \, C \, | \, C \, t \, | \, t \, C \, | \, \text{fix}(C) \]

Typing extended to evaluation contexts: \( \Gamma \vdash_{\Delta, \sigma} C : \tau \).
Typing extended to evaluation contexts: $\Gamma \vdash_{\Delta,\sigma} C : \tau$.

\[
\Gamma \vdash_{\Gamma,\tau} \tau - : \tau \quad \frac{\Gamma \vdash_{\Delta,\sigma} C : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash u : \tau_1}{\Gamma \vdash_{\Delta,\sigma} C u : \tau_2} \quad \ldots
\]
Given a type $\tau$, a typing context $\Gamma$ and terms $t, t' \in \text{PCF}_{\Gamma,\tau}$, contextual equivalence, written $\Gamma \vdash t \equiv_{\text{ctx}} t' : \tau$ is defined to hold if for all evaluation contexts $C$ such that $\cdot \vdash_{\Gamma,\tau} C : \gamma$, where $\gamma$ is $\text{nat}$ or $\text{bool}$, and for all values $v \in \text{PCF}_\gamma$,

$$C[t] \downarrow_\gamma v \iff C[t'] \downarrow_\gamma v.$$ 

When $\Gamma$ is the empty context, we simply write $t \equiv_{\text{ctx}} t' : \tau$ for $\cdot \vdash t \equiv_{\text{ctx}} t' : \tau$. 

PCF

INTRODUCING DENOTATIONAL SEMANTICS
THE AIMS OF DENOTATIONAL SEMANTICS

- a mapping of PCF types \( \tau \) to domains \([\tau]\);  
- a mapping of closed, well-typed PCF terms \( \vdash t : \tau \) to elements \([t] \in [\tau]\);  
- denotation of open terms will be continuous functions.
The aims of denotational semantics

- a mapping of PCF types \( \tau \) to domains \([\tau]\);
- a mapping of closed, well-typed PCF terms \( \vdash t : \tau \) to elements \([t] \in [\tau] \);
- denotation of open terms will be continuous functions.

Compositionality: \([t] = [t'] \implies [C[t]] = [C[t']] \).

Soundness: for any type \( \tau \), \( t \Downarrow_{\tau} v \implies [t] = [v] \).

Adequacy: for \( \gamma = \text{bool} \) or \( \text{nat} \), if \( t \in \text{PCF}_\gamma \) and \([t] = [v] \) then \( t \Downarrow_{\gamma} v \).
Proof principle: to show

\[ t_1 \cong_{\text{ctx}} t_2 : \tau \]

it suffices to establish

\[ [t_1] = [t_2] \in [\tau] \]
The power of denotational semantics

Proof principle: to show

\[ t_1 \cong_{\text{ctx}} t_2 : \tau \]

it suffices to establish

\[ [t_1] = [t_2] \in \llbracket \tau \rrbracket \]

\[ C[t_1] \downarrow_{\text{nat}} \nu \Rightarrow [C[t_1]] = [\nu] \] (soundness)

\[ \Rightarrow [C[t_2]] = [\nu] \] (compositionality on \([t_1] = [t_2]\))

\[ \Rightarrow C[t_2] \downarrow_{\text{nat}} \nu \] (adequacy)
Proof principle: to show

\[ t_1 \simeq_{\text{ctx}} t_2 : \tau \]

it suffices to establish

\[ [t_1] = [t_2] \in [\tau] \]

\[
C[t_1] \downarrow_{\text{nat}} v \Rightarrow [C[t_1]] = [v] \quad \text{(soundness)}
\]
\[
\Rightarrow [C[t_2]] = [v] \quad \text{(compositionality on } [t_1] = [t_2])
\]
\[
\Rightarrow C[t_2] \downarrow_{\text{nat}} v \quad \text{(adequacy)}
\]

and symmetrically for \( C[t_2] \downarrow_{\text{nat}} v \Rightarrow C[t_1] \downarrow_{\text{nat}} v \), and similarly for \textbf{bool}.\]
Proof principle: to show

\[ t_1 \simeq_{\text{ctx}} t_2 : \tau \]

it suffices to establish

\[ [t_1] = [t_2] \in [\tau] \]

Denotational equality is **sound**, but is it **complete**?
Does equality in the model imply contextual equivalence?
Proof principle: to show

\[ t_1 \equiv_{\text{ctx}} t_2 : \tau \]

it suffices to establish

\[ [t_1] = [t_2] \in [\tau] \]

Denotational equality is **sound**, but is it **complete**?
Does equality in the model imply contextual equivalence?

**Full abstraction.**
DENOTATIONAL SEMANTICS FOR PCF
DENOTATIONAL SEMANTICS FOR PCF

TYPES AND CONTEXTS
}\text{nat} \overset{\text{def}}{=} \mathbb{N}_ot \\
\text{bool} \overset{\text{def}}{=} \mathbb{B}_ot \\
[\tau \to \tau'] \overset{\text{def}}{=} [\tau] \to [\tau'] \\
\text{(function domain)}

\text{(flat domain)}
$$[\Gamma] \overset{\text{def}}{=} \prod_{x \in \text{dom}(\Gamma)} [\Gamma(x)] \quad (\Gamma\text{-environments})$$
Semantics of contexts

\[ [\Gamma] \overset{\text{def}}{=} \prod_{x \in \text{dom}(\Gamma)} [\Gamma(x)] \quad (\Gamma\text{-environments}) \]

- \([\cdot]\) = 1 (one element set)
- \([x: \tau] = (\{x\} \rightarrow [\tau]) \cong [\tau]\)
- \([x_1: \tau_1, \ldots, x_n: \tau_n] = [\tau_1] \times \cdots \times [\tau_n]\)
DENOTATIONAL SEMANTICS FOR PCF TERMS
To every typing judgement

$$\Gamma \vdash t : \tau$$

we associate a continuous function

$$[\Gamma \vdash t : \tau] : [\Gamma] \rightarrow [\tau]$$

between domains. In other words,

$$[-] : \text{PCF}_{\Gamma,\tau} \rightarrow [\Gamma] \rightarrow [\tau]$$
Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

\[
\text{succ} : \mathbb{N} \rightarrow \mathbb{N} \quad \text{pred} : \mathbb{N} \rightarrow \mathbb{N}
\]

\[
succ(n) = n + 1 \quad \text{pred}(n) = \begin{cases} 
0 & \text{undefined} \\
0 & \text{true} \\
n + 1 & \text{false}
\end{cases}
\]

\[
\text{zero?} : \mathbb{N} \rightarrow \mathbb{B} \quad q \quad \text{if} \quad b \quad \text{then} \quad t \quad \text{else} \quad t' \\
\text{def} = \begin{cases} 
0 & \text{true} \\
n + 1 & \text{false}
\end{cases}
\]

\[
\text{def} = \begin{cases} 
0 & \text{true} \\
n & \text{false}
\end{cases}
\]
**Denotation of operations on \( \mathbb{B} \) and \( \mathbb{N} \)**

\[
\text{succ}_\bot : \quad \mathbb{N}_\bot \rightarrow \mathbb{N}_\bot \\
\quad n \mapsto n + 1 \\
\quad \bot \mapsto \bot
\]

\[
\text{pred}_\bot : \quad \mathbb{N}_\bot \rightarrow \mathbb{N}_\bot \\
\quad 0 \mapsto \bot \\
\quad n + 1 \mapsto n \\
\quad \bot \mapsto \bot
\]

\[
\text{zero?}_\bot : \quad \mathbb{N}_\bot \rightarrow \mathbb{B}_\bot \\
\quad 0 \mapsto \text{true} \\
\quad n + 1 \mapsto \text{false} \\
\quad \bot \mapsto \bot
\]

\[
\text{if} \quad b \quad \text{then} \quad t \quad \text{else} \quad t' \\
\defeq \text{if}(b, t, q(t', \rho)) \in J(\tau)
\]
Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

\[
\begin{align*}
\text{[0]}(\rho) & \overset{\text{def}}{=} 0 & \in \mathbb{N}_\bot \\
\text{[true]}(\rho) & = \text{true} & \in \mathbb{B}_\bot \\
\text{[false]}(\rho) & = \text{false} & \in \mathbb{B}_\bot
\end{align*}
\]
Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

\[
\begin{align*}
[0](\rho) & \overset{\text{def}}{=} 0 \in \mathbb{N}_\perp \\
[\text{true}](\rho) & \overset{\text{def}}{=} \text{true} \in \mathbb{B}_\perp \\
[\text{false}](\rho) & \overset{\text{def}}{=} \text{false} \in \mathbb{B}_\perp \\
[succ(t)](\rho) & \overset{\text{def}}{=} \text{succ}_\perp([t](\rho)) \in \mathbb{N}_\perp \\
[pred(t)](\rho) & \overset{\text{def}}{=} \text{pred}_\perp([t](\rho)) \in \mathbb{N}_\perp \\
[\text{zero?}(t)](\rho) & \overset{\text{def}}{=} \text{zero?}_\perp([t](\rho)) \in \mathbb{B}_\perp
\end{align*}
\]

\[ [\text{succ}(t)] = \text{succ}_\perp \circ [t] \]
Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

\[
\begin{align*}
[0](\rho) & \overset{\text{def}}{=} 0 & \in \mathbb{N}_\perp \\
[\text{true}](\rho) & \overset{\text{def}}{=} \text{true} & \in \mathbb{B}_\perp \\
[\text{false}](\rho) & \overset{\text{def}}{=} \text{false} & \in \mathbb{B}_\perp \\
[\text{succ}(t)](\rho) & \overset{\text{def}}{=} \text{succ}_\perp([t](\rho)) & \in \mathbb{N}_\perp \\
[\text{pred}(t)](\rho) & \overset{\text{def}}{=} \text{pred}_\perp([t](\rho)) & \in \mathbb{N}_\perp \\
[\text{zero?}(t)](\rho) & \overset{\text{def}}{=} \text{zero?}_\perp([t](\rho)) & \in \mathbb{B}_\perp \\
[\text{if } b \text{ then } t \text{ else } t'] & \overset{\text{def}}{=} \text{if}([b](\rho), [t](\rho), [t'](\rho)) & \in [\tau] \\
[\text{if } b \text{ then } t \text{ else } t'] & = \text{if }\circ\langle [b], [t], [t'] \rangle
\end{align*}
\]
[x] (\rho) \overset{\text{def}}{=} \rho(x) \in \Gamma(x)
**Denotation of the \( \lambda \)-calculus operations**

\[
[x] (\rho) \overset{\text{def}}{=} \rho(x) \in \Gamma(x)
\]

\[
[t_1 \ t_2] (\rho) \overset{\text{def}}{=} ([t_1] (\rho)) ([t_2] (\rho))
\]

\[
[t_1 \ t_2] = \text{eval} \circ \langle [t_1], [t_2] \rangle
\]
Denotation of the $\lambda$-calculus operations

\[
\begin{align*}
[x] (\rho) & \overset{\text{def}}{=} \rho(x) \quad \in [\Gamma(x)] \\
[t_1 \ t_2] (\rho) & \overset{\text{def}}{=} ([t_1] (\rho)) ([t_2] (\rho)) \\
[\text{fun } x: \tau. \ t] (\rho) & \overset{\text{def}}{=} \lambda d \in [\tau]. [t] (\rho, d)
\end{align*}
\]

\[
[\text{fun } x: \tau. \ t] = \text{cur}([t])
\]
\[ [\text{fix } f](\rho) \overset{\text{def}}{=} \text{fix}(\lfloor f \rfloor(\rho)) \]
For any PCF term $t$ such that $\Gamma \vdash t : \tau$, the object $[t]$ is well-defined and a continuous function $[t] : [\Gamma] \rightarrow \tau$. 
For any PCF term $t$ such that $\Gamma \vdash t : \tau$, the object $[t]$ is well-defined and a continuous function $[t] : [\Gamma] \to \tau$.

If $t \in \text{PCF}_\tau$: $[t] \in [\cdot] \to [\tau] = 1 \to [\tau] \cong [\tau]$
DENOTATIONAL SEMANTICS FOR PCF

COMPOSITIONALITY
Suppose $t, u \in \text{PCF}_{\Gamma, \tau}$, such that

$$[t] = [u] : [\Gamma] \to [\tau]$$

Suppose moreover that $C[\cdot]$ is a PCF context such that $\Gamma' \vdash_{\Gamma, \tau} C : \tau'$. Then

$$[C[t]] = [C[u]] : [\Gamma'] \to [\tau'].$$
If $\Gamma \vdash_{\Delta, \sigma} C : \tau$, then define $\llbracket C \rrbracket$ such that

$$\llbracket C \rrbracket : (\llbracket \Delta \rrbracket \to \llbracket \sigma \rrbracket) \to \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$$
If $\Gamma \vdash_{\Delta, \sigma} C : \tau$, then define $\llbracket C \rrbracket$ such that

$$\llbracket C \rrbracket : (\llbracket \Delta \rrbracket \to \llbracket \sigma \rrbracket) \to \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$$

$$\llbracket [t] \rrbracket (d)(\rho) = (\llbracket C \rrbracket (d)(\rho))(\llbracket t \rrbracket (\rho))$$

$$\vdots$$
If $\Gamma \vdash_{\Delta, \sigma} C : \tau$, then define $\llbracket C \rrbracket$ such that

$$\llbracket C \rrbracket : ([\Delta] \rightarrow [\sigma]) \rightarrow [\Gamma] \rightarrow [\tau]$$

$$[-](d) = d$$

$$\llbracket C \, t \rrbracket (d)(\rho) = ([C] (d)(\rho))(\llbracket t \rrbracket (\rho))$$

$$\vdots$$

If $\Gamma \vdash_{\Delta, \sigma} C : \tau$ and $\Delta \vdash t : \sigma$, then

$$\llbracket [C \, t] \rrbracket = \llbracket C \rrbracket ([t])$$
Assume

\[ \Gamma \vdash u : \sigma \]
\[ \Gamma, x : \sigma \vdash t : \tau \]

Then for all \( \rho \in \llbracket \Gamma \rrbracket \)

\[ \llbracket t[u/x] \rrbracket (\rho) = \llbracket t \rrbracket (\rho[x \mapsto \llbracket u \rrbracket (\rho))]. \]

In particular when \( \Gamma = \cdot, \llbracket t \rrbracket : \llbracket \sigma \rrbracket \rightarrow \llbracket \tau \rrbracket \) and

\[ \llbracket t[u/x] \rrbracket = \llbracket t \rrbracket (\llbracket u \rrbracket) \]
DENOTATIONAL SEMANTICS FOR PCF

SOUNDNESS
For all PCF types $\tau$ and all closed terms $t, v \in \text{PCF}_\tau$ with $v$ a value, if $t \downarrow_\tau v$ is derivable, then

$$[t] = [v] \in [\tau]$$
Relating Denotational and Operational Semantics
For any **closed** PCF term $t$ and value $v$ of **ground** type $\gamma \in \{\text{nat, bool}\}$

$$[t] = [v] \in [\gamma] \Rightarrow t \downarrow_\gamma v$$
REMINDER: ADEQUACY

For any closed PCF term $t$ and value $v$ of ground type $\gamma \in \{\text{nat, bool}\}$

$$[t] = [v] \in [\gamma] \Rightarrow t \downarrow_\gamma v$$

Adequacy does not hold at function types or for open terms
For any closed PCF term $t$ and value $v$ of ground type $\gamma \in \{\text{nat}, \text{bool}\}$

$$[t] = [v] \in [\gamma] \Rightarrow t \downarrow_\gamma v$$

Adequacy does not hold at function types or for open terms

$$[\text{fun } x: \tau. (\text{fun } y: \tau. y) \, x] \; = \; [\text{fun } x: \tau. \, x] : [\tau] \rightarrow [\tau]$$

but

$$\text{fun } x: \tau. (\text{fun } y: \tau. y) \, x \downarrow_{\tau \rightarrow \tau} \; \text{fun } x: \tau. \, x$$
Relating Denotational and Operational Semantics

Formal approximation relation
Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}, n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow^n n$ (same for booleans);
2. for any well-typed term $t$, $R([t], t)$;
HOW TO PROVE ADEQUACY

Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);  
2. for any well-typed term $t$, $R([t], t)$;

Assume $t, v \in \text{PCF}_{\text{nat}}$, $[t] = [v]$, and $v$ is a value.
Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);
2. for any well-typed term $t$, $R([t], t)$;

Assume $t, v \in \text{PCF}_{\text{nat}}$, $[t] = [v]$, and $v$ is a value.

Thus $v = n$ for some $n \in \mathbb{N}$, and $[v] = n$. 
Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);
2. for any well-typed term $t$, $R([t], t)$;

Assume $t, v \in \text{PCF}_{\text{nat}}$, $[t] = [v]$, and $v$ is a value.

Thus $v = n$ for some $n \in \mathbb{N}$, and $[v] = n$.

\[
[t] = [n] = n \\
\Rightarrow R(n, t) \\
\Rightarrow t \Downarrow n = v
\]
Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);
2. for any well-typed term $t$, $R(\llbracket t \rrbracket, t)$;

But at non-base types, adequacy does not hold.
HOW TO PROVE ADEQUACY

Proof idea: introduce a relation $R$ such that

1. if $t \in \text{PCF}_{\text{nat}}, n \in \mathbb{N}$, and $R(n, t)$, then $t \downarrow_n$ (same for booleans);
2. for any well-typed term $t$, $R([t], t)$;

But at non-base types, adequacy does not hold.

We must define a family of relations, tailored for each type: formal approximation

$$<_{\tau} \subseteq [\tau] \times \text{PCF}_{\tau}$$
FORMAL APPROXIMATION AT BASE TYPES

\[
\begin{align*}
    d \triangleleft_{\text{nat}} t \ & \overset{\text{def}}{\iff} (d \in \mathbb{N} \Rightarrow t \downarrow_{\text{nat}} d) \\
    d \triangleleft_{\text{bool}} t \ & \overset{\text{def}}{\iff} (d = \text{true} \Rightarrow t \downarrow_{\text{bool}} \text{true}) \\
    & \quad \land (d = \text{false} \Rightarrow t \downarrow_{\text{bool}} \text{false})
\end{align*}
\]
FORMAL APPROXIMATION AT BASE TYPES

\[\begin{align*}
\text{def } d \succeq_{\text{nat}} t & \iff (d \in \mathbb{N} \Rightarrow t \downarrow_{\text{nat}} d) \\
\text{def } d \succeq_{\text{bool}} t & \iff (d = \text{true} \Rightarrow t \downarrow_{\text{bool}} \text{true}) \\
& \quad \land (d = \text{false} \Rightarrow t \downarrow_{\text{bool}} \text{false})
\end{align*}\]

Exactly what we need to get 1.
**FORMAL APPROXIMATION AT BASE TYPES**

\[
\begin{align*}
d \triangleleft_{\text{nat}} t & \overset{\text{def}}{\iff} (d \in \mathbb{N} \Rightarrow t \downarrow_{\text{nat}} d) \\
d \triangleleft_{\text{bool}} t & \overset{\text{def}}{\iff} (d = \text{true} \Rightarrow t \downarrow_{\text{bool}} \text{true}) \\
& \quad \land (d = \text{false} \Rightarrow t \downarrow_{\text{bool}} \text{false})
\end{align*}
\]

Exactly what we need to get 1.

Note though that \( \bot \triangleleft_{\text{nat}} t \) for any \( t \in \text{PCF}_{\text{nat}} \).
1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans); 

2. for any well-typed term $t$, $R([t], t)$. 

1. if \( t \in \text{PCF}_{\text{nat}} \), \( n \in \mathbb{N} \), and \( R(n, t) \), then \( t \Downarrow n \) (same for booleans);

2. for any well-typed term \( t \), \( R(\llbracket t \rrbracket, t) \).
   
   2.1 By induction on (the typing derivation of) \( t \);
   
   2.2 we need to interpret each typing rule.
1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);

2. for any well-typed term $t$, $R([t], t)$.
   
   2.1 By induction on (the typing derivation of) $t$;
   2.2 we need to interpret each typing rule.
1. if $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans); ✓

2. for any well-typed term $t$, $R([t], t)$.
   
   2.1 By induction on (the typing derivation of) $t$;
   
   2.2 we need to interpret each typing rule.

\[
\begin{array}{c}
\text{APP} \\
\vdash t : \tau \rightarrow \tau' \\
\vdash u : \tau \\
\hline
\vdash t u : \tau'
\end{array}
\]

Assume $[u] \triangleright_{\tau} u$ and $[t] \triangleright_{\tau \rightarrow \tau'} t$, how do we get $[t u] = [t] ([u]) \triangleright_{\tau} t u$?
1. If $t \in \text{PCF}_{\text{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow n$ (same for booleans);

2. For any well-typed term $t$, $R([t], t)$.
   2.1 By induction on (the typing derivation of) $t$;
   2.2 We need to interpret each typing rule.

\[
\text{APP} \vdash t : \tau \rightarrow \tau' \quad \vdash u : \tau \\
\hline
\vdash t u : \tau'
\]

Assume $[u] \triangleleft_{\tau} u$ and $[t] \triangleleft_{\tau \rightarrow \tau'} t$, how do we get $[t u] = [t] ([u]) \triangleleft_{\tau} t u$?

Define

\[
d \triangleleft_{\tau \rightarrow \tau'} t \quad \overset{\text{def}}{=} \quad \forall e \in [\tau], u \in \text{PCF}_{\tau}. (e \triangleleft_{\tau} u \Rightarrow d(e) \triangleleft_{\tau'} t u)
\]
To prove Item 2, we need to talk about open terms.
FORMAL APPROXIMATION FOR OPEN TERMS

\[ \text{ABS} \quad \frac{\Gamma, x: \tau \vdash t : \tau'}{\Gamma \vdash \text{fun } x: \tau. t : \tau \rightarrow \tau'} \]

To prove Item 2, we need to talk about open terms.

\[ [t](\llbracket u \rrbracket) = \llbracket (t[u/x]) \rrbracket \quad \text{Semantic application} \approx \text{syntactic substitution} \]
To prove Item 2, we need to talk about open terms.

\[
[t] ([u]) = [(t[u/x])] \quad \text{Semantic application} \approx \text{syntactic substitution}
\]

**Fundamental property of formal approximation**

Given a term \( t \) such that \( \Gamma \vdash t : \tau \) for some \( \Gamma \) and \( \tau \), for any environment \( \rho \) and substitution \( \sigma \) such that \( \rho \triangleleft \Gamma \sigma \), we have \([t] (\rho) \triangleleft_\tau t[\sigma] \).
To prove Item 2, we need to talk about open terms.

\[ [t]([u]) = [(t[u/x])] \quad \text{Semantic application} \approx \text{syntactic substitution} \]

**Fundamental property of formal approximation**

Given a term \( t \) such that \( \Gamma \vdash t : \tau \) for some \( \Gamma \) and \( \tau \), for any environment \( \rho \) and substitution \( \sigma \) such that \( \rho \triangleleft_\Gamma \sigma \), we have \( [t](\rho) \triangleleft_\tau t[\sigma] \).

Parallel substitution: maps each \( x \in \text{dom}(\Gamma) \) to \( \sigma(x) \in \text{PCF}_{\Gamma(x)} \).
RELATING DENOTATIONAL AND OPERATIONAL SEMANTICS

PROOF OF THE FUNDAMENTAL PROPERTY OF FORMAL APPROXIMATION
1. The least element approximates any program: for any $\tau$ and $t \in \text{PCF}_\tau$, $\perp[\tau] \triangleleft_\tau t$;

2. the set $\{d \in [\tau] \mid d \triangleleft_\tau t\}$ is chain-closed;
1. The least element approximates any program: for any $\tau$ and $t \in \text{PCF}_\tau$, $\bot_{[\tau]} \triangleleft_\tau t$;

2. the set $\{d \in [\tau] \mid d \triangleleft_\tau t\}$ is chain-closed;

3. if $\forall v. t \Downarrow_\tau v \Rightarrow t' \Downarrow_\tau v$, and $d \triangleleft_\tau t$, then $d \triangleleft_\tau t'$.
RELATING DENOTATIONAL AND OPERATIONAL SEMANTICS

EXTENSIONALITY
Contextual preorder is the one-sided version of contextual equivalence: \( \Gamma \vdash t \leq_{\text{ctx}} t' : \tau \)
if for all \( C \) such that \( \cdot \vdash_{\Gamma,\tau} C : \gamma \) and for all values \( v \),

\[ C[t] \downarrow_{\gamma} v \Rightarrow C[t'] \downarrow_{\gamma} v. \]
**Characterizing formal approximation**

Contextual preorder is the one-sided version of contextual equivalence: \( \Gamma \vdash t \leq_{\text{ctx}} t' : \tau \) if for all \( C \) such that \( \vdash_{\Gamma, \tau} C : \gamma \) and for all values \( v \),

\[
C[t] \downarrow_{\gamma} v \Rightarrow C[t'] \downarrow_{\gamma} v.
\]

\[
\Gamma \vdash t \equiv_{\text{ctx}} t' : \tau \Leftrightarrow (\Gamma \vdash t \leq_{\text{ctx}} t' : \tau \land \Gamma \vdash t' \leq_{\text{ctx}} t : \tau)
\]
**Contextual preorder** is the one-sided version of contextual equivalence: \( \Gamma \vdash t \leq_{\text{ctx}} t' : \tau \)

if for all \( C \) such that \( \cdot \vdash_{\Gamma, \tau} C : \gamma \) and for all values \( v \),

\[
C[t] \downarrow_{\gamma} v \Rightarrow C[t'] \downarrow_{\gamma} v.
\]

\[
\Gamma \vdash t \equiv_{\text{ctx}} t' : \tau \Leftrightarrow (\Gamma \vdash t \leq_{\text{ctx}} t' : \tau \land \Gamma \vdash t' \leq_{\text{ctx}} t : \tau)
\]

It corresponds to formal approximation: for all PCF types \( \tau \) and closed terms \( t_1, t_2 \in \text{PCF}_\tau \)

\[
t_1 \leq_{\text{ctx}} t_2 : \tau \Leftrightarrow \llbracket t_1 \rrbracket \prec_{\tau} t_2.
\]
Lemma: Application Contexts

For contextual preorder between closed terms, applicative contexts are enough.
For contextual preorder between closed terms, applicative contexts are enough.

Let $t_1, t_2$ be closed terms of type $\tau$. Then $t_1 \leq_{ctx} t_2 : \tau$ if and only if, for every term $f : \tau \rightarrow \text{bool}$,

$$f t_1 \downarrow_{\text{bool}} \text{true} \Rightarrow f t_2 \downarrow_{\text{bool}} \text{true}.$$
For $\gamma = \text{bool}$ or $\text{nat}$, $t_1 \leq_{\text{ctx}} t_2 : \tau$ holds if and only if

$$\forall v. (t_1 \Downarrow_\gamma v \Rightarrow t_2 \Downarrow_\gamma v).$$
For $\gamma = \text{bool}$ or $\text{nat}$, $t_1 \leq_{\text{ctx}} t_2 : \tau$ holds if and only if

$$\forall v. (t_1 \downarrow_\gamma v \Rightarrow t_2 \downarrow_\gamma v).$$

At a function type $\tau \rightarrow \tau'$, $t_1 \leq_{\text{ctx}} t_2 : \tau \rightarrow \tau'$ holds if and only if

$$\forall t \in \text{PCF}_\tau . (t_1 \downarrow t \leq_{\text{ctx}} t_2 t : \tau').$$
FULL ABSTRACTION
FULL ABSTRACTION

FAILURE OF FULL ABSTRACTION
A denotational model is **fully abstract** if

\[ t_1 \cong_{\text{ctx}} t_2 : \tau \Rightarrow \llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket \in \llbracket \tau \rrbracket \]
A denotational model is **fully abstract** if

\[ t_1 \cong_{ctx} t_2 : \tau \implies \llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket \in \llbracket \tau \rrbracket \]

A form of **completeness** of semantic equivalence wrt. program equivalence.
A denotational model is **fully abstract** if

\[ t_1 \equiv_{\text{ctx}} t_2 : \tau \Rightarrow [t_1] = [t_2] \in \tau \]

A form of **completeness** of semantic equivalence wrt. program equivalence.

The domain model of PCF is *not* fully abstract.
The parallel or function $\text{por} : \mathbb{B}_\bot \times \mathbb{B}_\bot \rightarrow \mathbb{B}_\bot$ is defined as given by the following table:

<table>
<thead>
<tr>
<th>por</th>
<th>true</th>
<th>false</th>
<th>$\bot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>true</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>
The (left) sequential or function \( \text{or} : B_\perp \times B_\perp \rightarrow B_\perp \) is defined as

\[
\text{or} \overset{\text{def}}{=} \left[ \text{fun} \ x : \text{bool}. \ \text{fun} \ y : \text{bool}. \ \text{if} \ x \ \text{then} \ \text{true} \ \text{else} \ y \right]
\]

It is given by the following table:

<table>
<thead>
<tr>
<th>or</th>
<th>true</th>
<th>false</th>
<th>( \perp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>( \perp )</td>
</tr>
<tr>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>( \perp )</td>
</tr>
</tbody>
</table>
### Parallel vs Sequential OR

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>true</strong></td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td><strong>false</strong></td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td><strong>true</strong></td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>true</strong></td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td><strong>false</strong></td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>
## Parallel vs Sequential OR

<table>
<thead>
<tr>
<th>por</th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>true</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>or</th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

*or* is **sequential**, but *por* is **not**.
There is no closed PCF term

\[ t : \text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \]

satisfying

\[ [t] = \text{por} : \mathbb{B}_\bot \rightarrow \mathbb{B}_\bot \rightarrow \mathbb{B}_\bot. \]
The denotational model of PCF in domains and continuous functions is not fully abstract.
The denotational model of PCF in domains and continuous functions is not fully abstract. For well-chosen $T_{\text{true}}$ and $T_{\text{false}}$,

$$T_{\text{true}} \equiv_{\text{ctx}} T_{\text{false}} : (\text{bool} \to \text{bool} \to \text{bool}) \to \text{bool}$$

$$[T_{\text{true}}] \neq [T_{\text{false}}] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$
The denotational model of PCF in domains and continuous functions is not fully abstract.

For well-chosen $T_{\text{true}}$ and $T_{\text{false}}$,

$$T_{\text{true}} \equiv_{\text{ctx}} T_{\text{false}} : (\text{bool} \to \text{bool} \to \text{bool}) \to \text{bool}$$

$$[T_{\text{true}}] \neq [T_{\text{false}}] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$

Idea:

- for all $f \in PCF_{\text{bool} \to \text{bool} \to \text{bool}}$, ensure $Tb \ f \uparrow_{\text{bool}}$
- but $[Tb] \ (\text{por}) = [b]$. 
$T_b \overset{\text{def}}{=} \text{fun } f : \text{bool } \to (\text{bool } \to \text{bool}).$
\[
\begin{align*}
&\quad \text{if}(f \text{ true } \Omega_{\text{bool}}) \text{ then } \\
&\quad \quad \text{if } (f \Omega_{\text{bool}} \text{ true}) \text{ then } \\
&\quad \quad \quad \text{if } (f \text{ false false}) \text{ then } \Omega_{\text{bool}} \text{ else } b \\
&\quad \quad \text{else } \Omega_{\text{bool}} \\
&\quad \text{else } \Omega_{\text{bool}}
\end{align*}
\]
FULL ABSTRACTION
BEYOND FULL ABSTRACTION FAILURE
• PCF is not expressive enough to present the model?
• The model does not adequately capture PCF?
• Contexts are too weak: they do not distinguish enough programs?
Γ ⊢ t : τ

Γ ⊢ t₁ : τ  Γ ⊢ t₂ : τ
Γ ⊢ por(t₁, t₂) : τ

t ↓ᵣ v

spar(t₁, t₂) ↓ᵣ bool true

spar(t₁, t₂) ↓ᵣ bool false

spar(t₁, t₂) ↓ᵣ bool true

spar(t₁, t₂) ↓ᵣ bool true
If we extend the semantics of PCF to PCF+$\text{por}$ with

$$[\text{por}] = \text{por}$$

the resulting denotational semantics is fully abstract.
If we extend the semantics of PCF to PCF+por with

\[ \text{[por]} = \text{por} \]

the resulting denotational semantics is fully abstract...

but is PCF+por still a reasonable model of programming language?
Fully abstract semantics for PCF

- first step: dl-domains & stable functions $\rightarrow$ no por any more, but still not fully abstract...
- only proper answers in the late 90s (!): logical relations and game semantics
Fully abstract semantics for PCF

• first step: dI-domains & stable functions → no *por* any more, but still not fully abstract...
• only proper answers in the late 90s (!): logical relations and game semantics

Real languages have *effects*

• If you add effects (references, control flow...) to a language, contexts become *much more* expressive.
• Full abstraction becomes different: somewhat easier... but is contextual equivalence still a reasonable idea?
WHERE TO GO FROM HERE?
Source of a very rich literature:

- linear logic
- logical relations
- game semantics
- bisimulations techniques
- …
Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)
Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)

Interpret:

- a type $\tau$ as an object in a category;
- a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[t] : [\Gamma] \to [\tau]$. 
CATEGORICAL SEMANTICS

Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)

Interpret:

- a type $\tau$ as an object in a category;
- a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[t] : [\Gamma] \rightarrow [\tau]$.

Example: $\lambda$-calculus $\rightarrow$ cartesian closed categories
OCaml’s ADT:

It is a **fixed point equation**! We can use domain theory to solve it.
Effects: control flow (errors), mutability/state, input-output...
An important aspect of programming languages!
Effects: control flow (errors), mutability/state, input-output...
An important aspect of programming languages!

Modelled as a monad $T$ (example: $T(A) \overset{\text{def}}{=} (A \times \text{State})^{\text{State}}$)
Effects: control flow (errors), mutability/state, input-output...
An important aspect of programming languages!

Modelled as a monad $T$ (example: $T(A) \overset{\text{def}}{=} (A \times \text{State})^{\text{State}}$)

Denotation of a computation: $[\Gamma] \rightarrow T([\tau])$
Easter: *axiomatic semantic* (Hoare Logic and Model Checking)
Easter: *axiomatic semantic* (Hoare Logic and Model Checking)

In the end, the most interesting aspects of semantics is in the *interaction* between different approaches.