Preliminary Concepts (1)

Inductive Definitions, Structural Induction, and Denotational Semantics

Woosuk Lee

CSE 6049 Program Analysis

Hanyang University, Korea

Some slides are borrowed from https://github.com/prosyslab-classroom/is593-2020-spring/blob/master/slides/lecture2.pdf

Inductive Definitions

Inductive Definitions

- Inductive definition (induction) is widely used in the study of programming languages and computer science in general: e.g.,
  - The syntax and semantics of programming languages
  - Data structures (e.g., lists, trees, graphs)
- Induction is a technique for formally defining a set:
  - The set is defined in terms of itself.
  - The only way of defining an infinite set by a finite means.

Examples of Inductive Definitions

- Definition of linked lists:
  - The empty list is a linked list.
  - A single node followed by a linked list is a linked list
- Definition of binary trees
  - The empty tree is a binary tree.
  - A node with two children that are binary trees is a binary tree.

Inference Rules

- An inference rule is of the form:

$$\frac{A}{B}$$

  - A: hypothesis (antecedent)
  - B: conclusion (consequent)
  - "if A is true then B is also true".
  - $\overline{B}$: axiom (inference rule without hypothesis)

The hypothesis may contain multiple statements:

$$\frac{A \quad B}{C}$$

"If both A and B are true then so is C".

Example

Suppose we want to define a set S of natural numbers which are multiples of 3.

The set S is defined as inference rules as follows:

Definition (S)

$$\frac{}{0 \in S} \qquad \frac{n \in S}{(n+3) \in S}$$

Interpret the rules as follows:

"A natural number n is in S iff $n \in S$ can be derived from the axiom by applying the inference rules finitely many times"

For example, $3 \in S$ because we can find a "proof/derivation tree":

$$\cfrac{\cfrac{}{0 \in S}\ \text{the axiom}}{3 \in S}\ \text{the second rule}$$

but $1, 2, 4, \cdots \notin S$ because we cannot find proofs. Note that this interpretation enforces that S is the smallest set closed under the inference rules.

Inference Rules

- What set is defined by the following inductive rules?

$$\frac{}{\boldsymbol{3}} \qquad \frac{\boldsymbol{x} \quad \boldsymbol{y}}{\boldsymbol{x + y}}$$

- What set is defined by the following inductive rules?

$$\frac{}{()} \qquad \frac{\boldsymbol{x}}{(\boldsymbol{x})} \qquad \frac{\boldsymbol{x} \quad \boldsymbol{y}}{\boldsymbol{xy}}$$

Inference Rules

- Define the following set as rules of inference:

$$S = \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, ...\}$$

- Define the following set as rules of inference:

$$S = \{a^n b^{n+1} \mid n \in \mathbb{N}\}$$

Natural Numbers

The set of natural numbers:

$$\mathbb{N} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots\}$$

is inductively defined:

$$\frac{}{\mathbf{0}} \qquad \frac{\boldsymbol{n}}{\boldsymbol{n} + \mathbf{1}}$$

The inference rules can be expressed by a grammar:

$$\boldsymbol{n} \to \mathbf{0} \mid \boldsymbol{n} + \mathbf{1}$$

Interpretation:

- 0 is a natural number.
- If $\boldsymbol{n}$ is a natural number then so is $\boldsymbol{n} + \mathbf{1}$.

Strings

The set of strings over alphabet $\{\texttt{a}, \ldots, \texttt{z}\}$, e.g., ϵ, $\texttt{a}$, $\texttt{b}$, $\ldots$, $\texttt{z}$, $\texttt{aa}$, $\texttt{ab}$, $\ldots$, $\texttt{az}$, $\texttt{ba}$, $\ldots$ $\texttt{az}$, $\texttt{aaa}$, $\ldots$, $\texttt{zzz}$, and so on. Inference rules:

$$\frac{}{\epsilon} \qquad \frac{\alpha}{\texttt{a}\alpha} \qquad \frac{\alpha}{\texttt{b}\alpha} \quad \cdots \qquad \frac{\alpha}{\texttt{z}\alpha}$$

or simply,

$$\frac{}{\epsilon} \qquad \frac{\alpha}{x\alpha}\ x \in \{\texttt{a}, \ldots, \texttt{z}\}$$

In grammar:

$$\begin{array}{rcl} \alpha & \rightarrow & \epsilon \\ & \mid & x\alpha \quad (x \in \{\texttt{a}, \ldots, \texttt{z}\}) \end{array}$$

Boolean Values

The set of boolean values:

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

If a set is finite, just enumerate all of its elements by axioms:

$$\frac{}{true} \qquad \frac{}{false}$$

In grammar:

$$b \rightarrow true \mid false$$

Lists

Examples of lists of integers:

1. $\textbf{nil}$
2. $\mathbf{14} \cdot \textbf{nil}$
3. $\mathbf{3} \cdot \mathbf{14} \cdot \textbf{nil}$
4. $\mathbf{-7} \cdot \mathbf{3} \cdot \mathbf{14} \cdot \textbf{nil}$

Inference rules:

$$\frac{}{\textbf{nil}} \qquad \frac{\boldsymbol{l}}{\boldsymbol{n} \cdot \boldsymbol{l}}\ \boldsymbol{n} \in \mathbb{Z}$$

In grammar:

$$\begin{array}{rcl} \boldsymbol{l} & \to & \textbf{nil} \\ & \mid & \boldsymbol{n} \cdot \boldsymbol{l} \quad (\boldsymbol{n} \in \mathbb{Z}) \end{array}$$

Lists

A proof that $\mathbf{-7 \cdot 3 \cdot 14 \cdot nil}$ is a list of integers:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\textbf{nil}} \quad \mathbf{14} \in \mathbb{Z}}{\mathbf{14} \cdot \textbf{nil}} \quad \mathbf{3} \in \mathbb{Z}}{\mathbf{3} \cdot \mathbf{14} \cdot \textbf{nil}} \quad \mathbf{-7} \in \mathbb{Z}}{\mathbf{-7} \cdot \mathbf{3} \cdot \mathbf{14} \cdot \textbf{nil}}$$

The proof tree is also called *derivation tree* or *deduction tree*.

Binary Trees

Binary tree examples: $\mathbf{1}$, $(\mathbf{1}, \textbf{nil})$, $(\mathbf{1}, \mathbf{2})$, $((\mathbf{1}, \mathbf{2}), \textbf{nil})$, $((\mathbf{1}, \mathbf{2}), (\mathbf{3}, \mathbf{4}))$.
Inference rules:

$$\frac{}{\boldsymbol{n}}\ \boldsymbol{n} \in \mathbb{Z} \qquad \frac{\boldsymbol{t}}{(\boldsymbol{t}, \textbf{nil})} \qquad \frac{\boldsymbol{t}}{(\textbf{nil}, \boldsymbol{t})} \qquad \frac{\boldsymbol{t_1} \qquad \boldsymbol{t_2}}{(\boldsymbol{t_1}, \boldsymbol{t_2})}$$

In grammar:

$$\begin{array}{rcll} \boldsymbol{t} & \rightarrow & \boldsymbol{n} & (\boldsymbol{n} \in \mathbb{Z}) \\ & \mid & (\boldsymbol{t}, \textbf{nil}) & \\ & \mid & (\textbf{nil}, \boldsymbol{t}) & \\ & \mid & (\boldsymbol{t}, \boldsymbol{t}) & \end{array}$$

A proof that $((\mathbf{1}, \mathbf{2}), (\mathbf{3}, \textbf{nil}))$ is a binary tree:

$$\frac{\dfrac{\overline{\mathbf{1}} \qquad \overline{\mathbf{2}}}{(\mathbf{1}, \mathbf{2})} \qquad \dfrac{\overline{\mathbf{3}}}{(\mathbf{3}, \textbf{nil})}}{((\mathbf{1}, \mathbf{2}), (\mathbf{3}, \textbf{nil}))}$$

Expressions

Expression examples: $\mathbf{2}$, $-\mathbf{2}$, $\mathbf{1}+\mathbf{2}$, $\mathbf{1}+(\mathbf{2}*(-\mathbf{3}))$, etc.
Inference rules:

$$\frac{}{\boldsymbol{n}}\ \boldsymbol{n}\in\mathbb{Z} \qquad \frac{\boldsymbol{e}}{-\boldsymbol{e}} \qquad \frac{\boldsymbol{e_1}\quad \boldsymbol{e_2}}{\boldsymbol{e_1}+\boldsymbol{e_2}} \qquad \frac{\boldsymbol{e_1}\quad \boldsymbol{e_2}}{\boldsymbol{e_1}*\boldsymbol{e_2}} \qquad \frac{\boldsymbol{e}}{(\boldsymbol{e})}$$

In grammar:

$$\begin{array}{rcll} \boldsymbol{e} & \rightarrow & \boldsymbol{n} & (\boldsymbol{n}\in\mathbb{Z}) \\ & \mid & -\boldsymbol{e} & \\ & \mid & \boldsymbol{e}+\boldsymbol{e} & \\ & \mid & \boldsymbol{e}*\boldsymbol{e} & \\ & \mid & (\boldsymbol{e}) & \end{array}$$

Example:

$$\dfrac{\dfrac{}{\mathbf{1}} \quad \dfrac{\dfrac{\dfrac{}{\mathbf{2}} \quad \dfrac{\dfrac{\dfrac{}{\mathbf{3}}}{-\mathbf{3}}}{(-\mathbf{3})}}{\mathbf{2}*(-\mathbf{3})}}{(\mathbf{2}*(-\mathbf{3}))}}{\mathbf{1}+(\mathbf{2}*(-\mathbf{3}))}$$

Structural Induction

Structural Induction

A technique for proving properties about inductively defined sets.

To prove that a proposition $P(s)$ is true for all structures s, prove the following:

1. (Base case) P is true on simple structures (those without substructures)
2. (Inductive case) If P is true on the substructures of s, then it is true on s itself. The assumption is called *induction hypothesis (I.H.)*.

Example

Let S be the set defined by the following inference rules:

$$\frac{}{3} \qquad \frac{x \quad y}{x + y}$$

Prove that for all $x \in S$, x is divisible by 3.
Proof. By structural induction.

- (Base case) The base case is when x is 3. Obviously, x is divisible by 3.
- (Inductive case) The induction hypothesis (I.H.) is

  $$x \text{ is divisible by } 3, \qquad y \text{ is divisible by } 3.$$

  Let $x = 3k_1$ and $y = 3k_2$. Using I.H., we derive

  $$x + y \text{ is divisible by } 3$$

  as follows:

  $$\begin{array}{rcll} x + y & = & 3k_1 + 3k_2 & \cdots \text{by I.H.} \\ & = & 3(k_1 + k_2) & \end{array}$$

□

Example

Let T be the set of binary trees:

$$\frac{}{\textbf{leaf}} \qquad \frac{t_1 \quad t_2}{(n, t_1, t_2)}\ n \in \mathbb{Z}$$

Prove that for all such trees, the number of leaves is always one more than the number of internal nodes.

Proof. Restate the claim more formally:

$$\text{If } t \in T \text{ then } l(t) = i(t) + 1$$

where $l(t)$ and $i(t)$ denote the number of leaves and internal nodes, respectively:

$$\begin{array}{rcl} l(\textbf{leaf}) & = & 1 \\ l(n, t_1, t_2) & = & l(t_1) + l(t_2) \end{array} \qquad \begin{array}{rcl} i(\textbf{leaf}) & = & 0 \\ i(n, t_1, t_2) & = & i(t_1) + i(t_2) + 1 \end{array}$$

We prove it by structural induction:

- (Base case): The base case is when $t = \textbf{leaf}$, where $l(t) = 1$ and $i(t) = 0$.
- (Inductive case): The induction hypothesis:

  $$l(t_1) = i(t_1) + 1, \qquad l(t_2) = i(t_2) + 1$$

  Using I.H., we prove $l((n, t_1, t_2)) = i((n, t_1, t_2)) + 1$:

  $$\begin{array}{rcll} l((n, t_1, t_2)) & = & l(t_1) + l(t_2) & \text{definition of of } l \\ & = & i(t_1) + 1 + i(t_2) + 1 & \text{by induction hypothesis} \\ & = & i(n, t_1, t_2) + 1 & \text{definition of } i \end{array}$$

From now on

See how to define a programming language

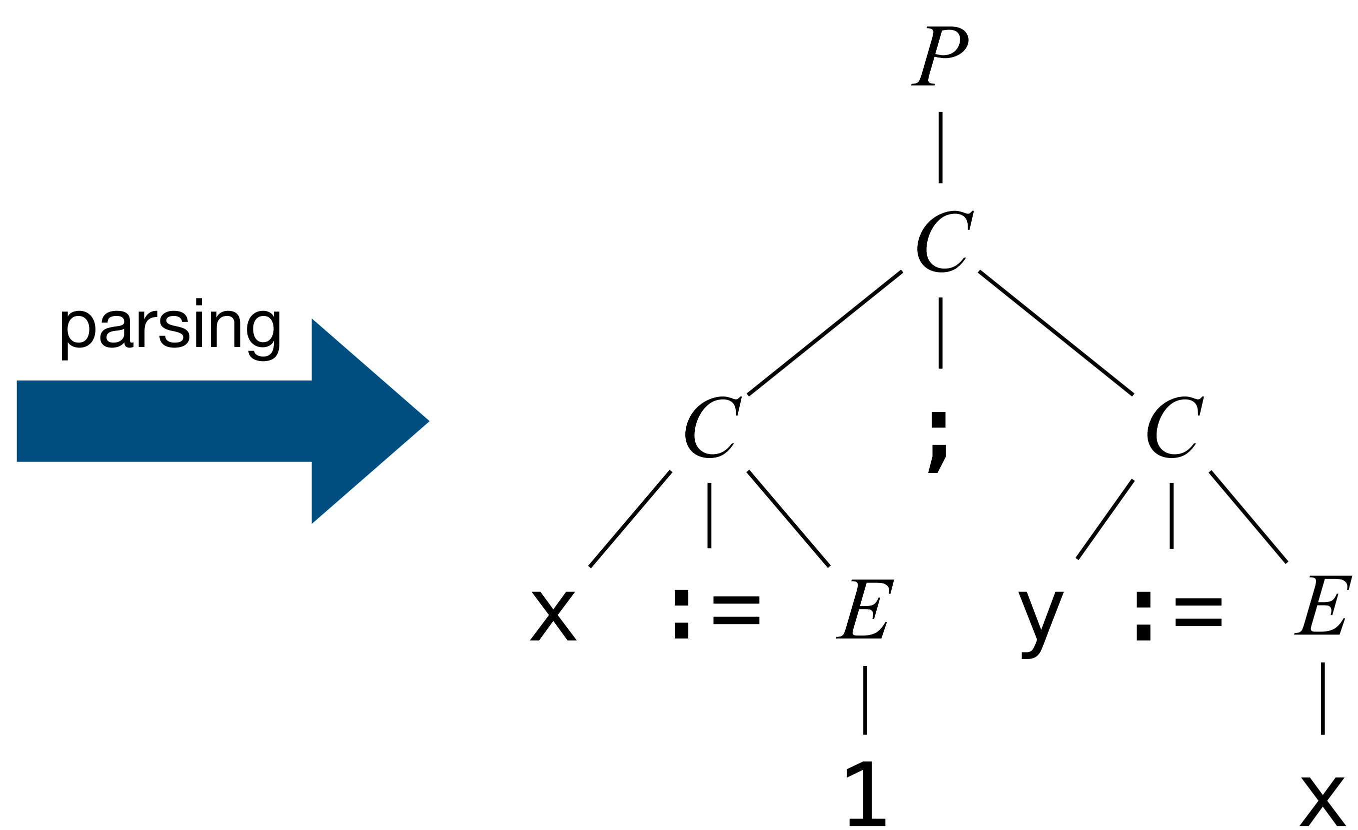
- A programming language = Syntax + Semantics
- Both are inductively defined.

Syntax

Syntax

- A grammar specifying how programs should look like (grammatical structures)
- Parsing : constructing an abstract syntax tree from a text

```
x := 1; y := x
```



Grammars for Expressions and Programs

- Expressions

$$
\begin{array}{rcll}
E & \rightarrow & n & (n \in \mathbb{Z}) \\
  & | & E \texttt{ + } E & \\
  & | & \texttt{-}\, E &
\end{array}
$$

terminal

non-terminal

terminal

terminal

- Simple program commands

$$
\begin{array}{rcl}
C & \rightarrow & \texttt{skip} \\
  & | & x \texttt{ := } E \\
  & | & \texttt{if}\ E\ C\ C \\
  & | & C\ ;\ C
\end{array}
$$

Grammars for Expressions and Programs (another version)

- Expressions

$$
\begin{array}{rcll}
E & \rightarrow & n & (n \in \mathbb{Z}) \\
 & | & \texttt{+}\ E\ E & \\
 & | & \texttt{-}\ E &
\end{array}
$$

Whatever terminals you want !

- Simple programs

$$
\begin{array}{rcl}
C & \rightarrow & \& \\
 & | & \texttt{=}\ x\ E \\
 & | & \texttt{?}\ E\ C\ C \\
 & | & \texttt{;}\ C\ C
\end{array}
$$

Abstract vs. Concrete Syntax

- Abstract syntax
  - Tree structure (2D) independent of any particular representation and encoding
- Concrete syntax
  - Source text (1D)
- E.g., concrete syntax includes features like parentheses (for grouping) or commas (for lists) which are not included in the abstract syntax

Abstract vs. Concrete Syntax

- Which one of the followings is `- 1 + 2`?
  - $\langle\langle$`-1`$\rangle$ `+ 2`$\rangle$ `or` `-`$\langle$`1 + 2`$\rangle$
- Cannot answer with :

$$
\begin{array}{rcll}
E & \rightarrow & n & (n \in \mathbb{Z}) \\
  & \mid & E \;\texttt{+}\; E & \\
  & \mid & \texttt{-}\; E &
\end{array}
$$

- Can answer with :

$$
\begin{array}{rcll}
E & \rightarrow & n & (n \in \mathbb{Z}) \\
  & \mid & E \;\texttt{+}\; E & \\
  & \mid & \texttt{-}\; F & \\
F & \rightarrow & n & \\
  & \mid & \texttt{(}\; E \;\texttt{)} &
\end{array}
$$

Abstract vs. Concrete Syntax

- Parsers convert concrete syntax into abstract syntax and have to deal with ambiguity
  - e.g., associativity and precedence
- From now on, a "program" refers to its abstract syntax.

Denotational Semantics

Semantics

- About what a program means
- What is the meaning of a program "1 + 2" ?
  - Meaning = what it "denotes": "3"
    (**Denotational semantics**)
  - Meaning = how to compute the result: "add 1 into 2 and get 3"
    (**Operational semantics**)

    …

Different approaches for different purposes and languages

Denotational Semantics

- Mathematical meaning of a program (no machine states or transitions)
- Program semantics is a function from input states to output states
- The semantics of a program is determined by that of each component (i.e., compositional)

Semantics of a Simple Language (WHILE)

$$
\begin{array}{rcll}
C & \to & \texttt{skip} & \\
 & | & x := E & \\
 & | & \texttt{if}\ E\ C\ C & \\
 & | & C; C & \\
 & | & \texttt{while}\ E\ C & \\
E & \to & n & (n \in \mathbb{Z}) \\
 & | & x & \\
 & | & E\ \texttt{+}\ E & \\
 & | & \texttt{-}\ E &
\end{array}
$$

- The semantics of C is a function from memories to memories
- Memory = Function from memory locations to values

Semantic Domain

- A set of objects used to define program semantics (i.e., semantic objects)

$$\begin{array}{rcl} M \in \mathit{Memory} & = & \mathit{Var} \to \mathit{Value} \\ z \in \mathit{Value} & = & \mathbb{Z} \\ x \in \mathit{Var} & = & \mathit{ProgramVariable} \end{array}$$

- Meaning of commands $[\![C]\!] \in \mathit{Memory} \to \mathit{Memory}$

- Meaning of expressions $[\![E]\!] \in \mathit{Memory} \to \mathbb{Z}$

Denotational Semantics of the Language

$$
\begin{array}{rcl}
[\![\texttt{skip}]\!]\, M & = & M \\
[\![x\ \texttt{:=}\ E]\!]\, M & = & M\{x \mapsto [\![E]\!]\, M\} \\
[\![\texttt{if}\ E\ C_1\ C_2]\!]\, M & = & \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![C]\!]M\ \mathit{else}\ [\![C_2]\!]M \\
[\![C_1\ ;\ C_2]\!]\, M & = & [\![C_2]\!]\,([\![C_1]\!]\, M) \\
[\![n]\!]\, M & = & n \\
[\![E_1\ \texttt{+}\ E_2]\!]\, M & = & ([\![E_1]\!]\, M) + ([\![E_2]\!]\, M) \\
[\![\texttt{-}\ E]\!]\, M & = & -([\![E]\!]\, M)
\end{array}
$$

- E.g., $[\![\texttt{x:=7;y:=3}]\!]\{\} = \{\texttt{x} \mapsto 7, \texttt{y} \mapsto 3\}$
- Compositional! (i.e., the semantics of a program is determined by its sub-components)

Semantics of Loops

- The semantics of

$$\texttt{while}\ E\ C$$

$$\begin{aligned}&[\![\texttt{while}\ E\ C]\!]M \\ &= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M\end{aligned}$$

- Is it compositional?

Semantics of Loops

- The semantics of

$$\texttt{while}\ E\ C$$

$$\begin{aligned}&[\![\texttt{while}\ E\ C]\!]M\\&= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M\end{aligned}$$

- Is it compositional?
  ☞ No! Not a definition but just an **equation**

Semantics of Loops

$[\![\texttt{while}\ E\ C]\!]M$

$= \mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M$

How to denote functions:

$\lambda x.\ \text{function body}$

where x is a parameter

e.g., $\underline{\lambda x.\ x+1}$

$[\![\texttt{while}\ E\ C]\!] =$

$\lambda M.\mathit{if}\ [\![E]\!]M \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ M.$

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

where $$F_{E,C}(X) = \begin{cases} X([\![C]\!](m)) & ([\![E]\!](m) \neq 0) \\ m & (\mathit{otherwise}) \end{cases}$$

Semantics of Loops

- Semantics of a loop: a solution of this equation

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

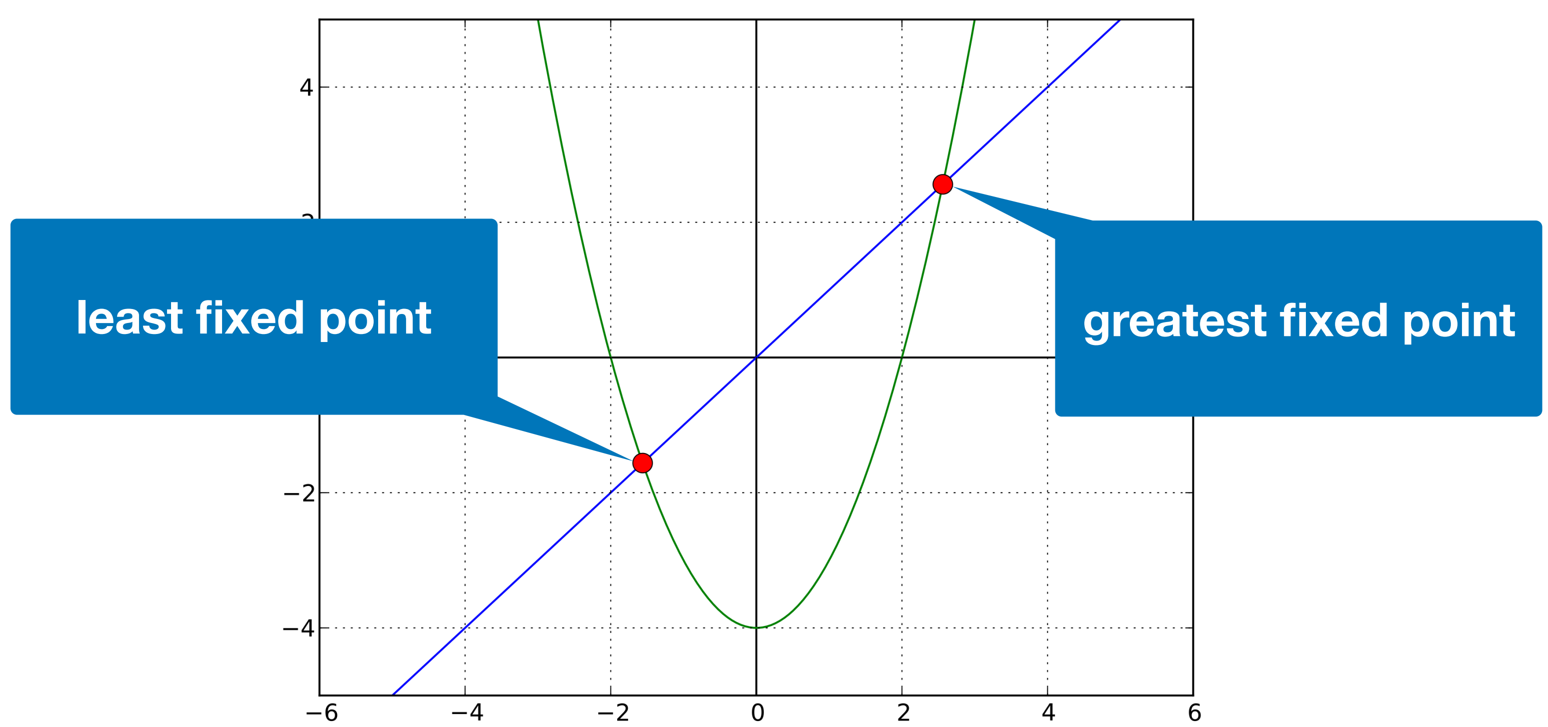
- Solution: a **fixed point** of $F_{E,C}$

Fixpoint?

$$\text{fix}F = X \text{ such that } F(X) = X$$

$f(x) = x^2 - 4$

lfp



gfp

*https://en.wikipedia.org/wiki/Least_fixed_point

Semantics of Loops

- Semantics of a loop: a solution of this equation

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

- Solution: a fixed point of $F_{E,C}$

$(Memory \to Memroy) \to (Memory \to Memroy)$

$$[\![\texttt{while}\ E\ C]\!] = \text{fix}\ F_{E,C}$$

$Memory \to Memory$

$$F_{E,C}(X) = \begin{cases} X([\![C]\!](m)) & ([\![E]\!](m) \neq 0) \\ m & (otherwise) \end{cases}$$

- Compositional ($[\![\texttt{while}\ E\ C]\!]$ is defined using $[\![E]\!]$, $[\![C]\!]$)

Exercise

- "Computer science is full of fix points."
  Inductively defined thing = a least fix point:

- `N = {0}` $\cup$ `{n+1|n` $\in$ `N}`

$$N = \mathit{fix} \lambda X.\{0\} \cup \{n + 1 | n \in X\}$$

- `list = {nil}` $\cup$ `{(0,l)|l`$\in$ `list}`

$$\mathit{list} = \mathit{fix} \lambda X.\{\texttt{nil}\} \cup \{(0, l) | l \in X\}$$

Exercise

- `reach(N) = N ∪ reach(next(N))`

$$reach = \mathit{fix} \lambda f.(\lambda N.N \cup f(next(N)))$$

- `fac(n) = if n=0? 1 : n*fac(n-1)`

$$fac = \mathit{fix} \lambda f.(\lambda n.\mathit{if}\ n = 0?\, 1 :\ n \times f(n-1))$$

Questions

- Does a solution of the semantic equation always exist?
- If exists, is it unique?
- How to compute it?

Domain Theory

- Semantics of a program is an element of a domain called CPO (complete partial ordered set)
- Semantics of a program is the least fix point of a continuous function.
- Established by Dana Scott in 1970s
    - Outline of a Mathematical Theory of Computation, Dana Scott
    - Mathematical Concepts in Programming Language Semantics, Dana Scott
    - Domains and Logics, Dana Scott

Intuitions behind Domain Theory

- Goal: giving a mathematical meaning to each program

- Problem: what is the meaning of the following program?

  $\texttt{while}\ (1)\ \{x := x + 1\}$ ☞ ***never terminates!***

- Need something to represent an *undefined output* (written $\perp$), i.e., the result of a computation that never ends.

Intuitions behind Domain Theory

- There is an ordering between elements of the domains of computation.
  - e.g., Type `int` is more specific than type `double`
  - e.g., any value is more informative than $\perp$ (i.e., no information)
- The higher an element is within the order, the more information it contains.

Partial Order

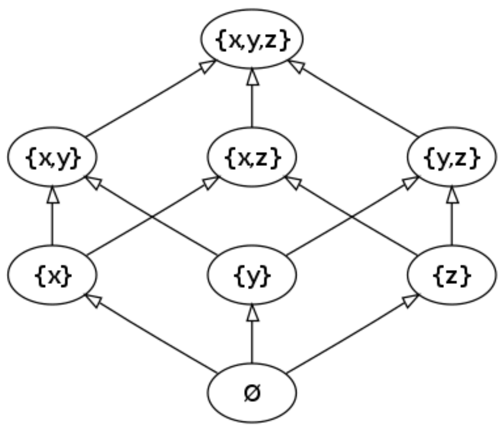
Definition (Partial Order). A binary relation $\sqsubseteq$ is a **partial order** on a set D if it has:

1. reflexivity: $a \sqsubseteq a$ for all $a \in D$
2. Antisymmetry: $a \sqsubseteq b$ and $b \sqsubseteq a$ implies $a = b$
3. Transitivity: $a \sqsubseteq b$ and $b \sqsubseteq c$ implies $a \sqsubseteq c$

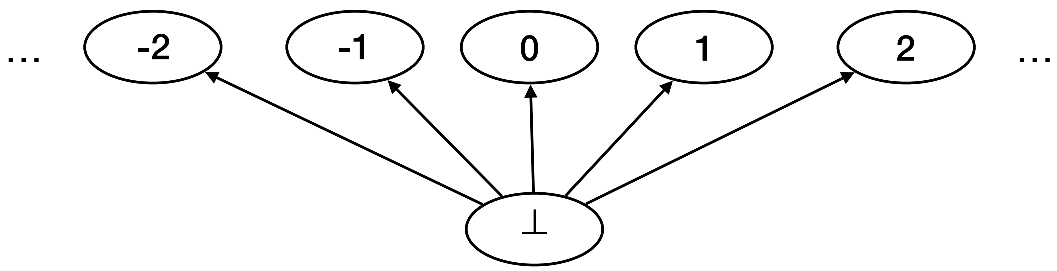
A set D with a partial order $\sqsubseteq$ is called a **partially ordered set** $(D, \sqsubseteq)$, or simply **poset**.

Powerset: {{}, {x}, {y}, {z}, {x,y}, {y,z}, {x,z}, {x,y,z}}

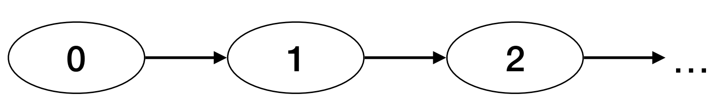
- Example 1: $(\wp(\{x, y, z\}), \subseteq)$



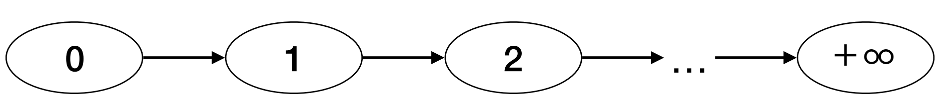
- Example 2: $(\mathbb{Z}_\perp, \sqsubseteq)$



- Example 3: $(\mathbb{N}, \leq)$



- Example 4: $(\mathbb{N} + \{+\infty\}, \leq)$



Graphical representations of partial orders are called Hasse diagrams.

Least Upper Bound

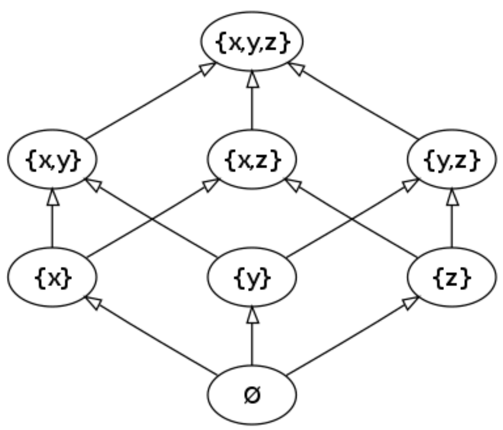
Definition (Least Upper Bound). For a partial ordered set $(D, \sqsubseteq)$ and subset $X \subseteq D$, $d \in D$ is an **upper bound** of X iff

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

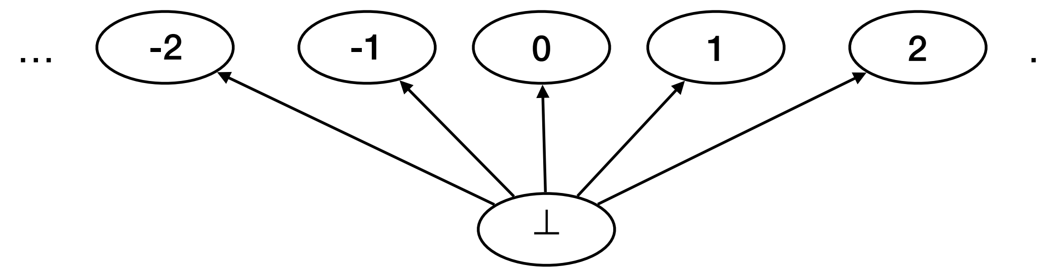
An upper bound d is the **least upper bound** of X iff for all upper bounds y of X, $d \sqsubseteq y$. The least upper bound of X is denoted by $\bigsqcup X$.

Intuition: union of multiple pieces of information
e.g., Set union (∪)

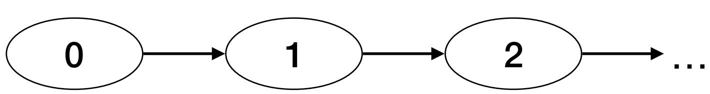
- Example 1: $(\wp(\{x, y, z\}), \subseteq)$



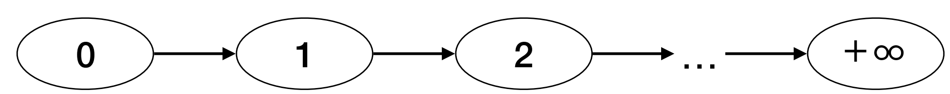
- Example 2: $(\mathbb{Z}_\perp, \sqsubseteq)$



- Example 3: $(\mathbb{N}, \leq)$



- Example 4: $(\mathbb{N} + \{+\infty\}, \leq)$

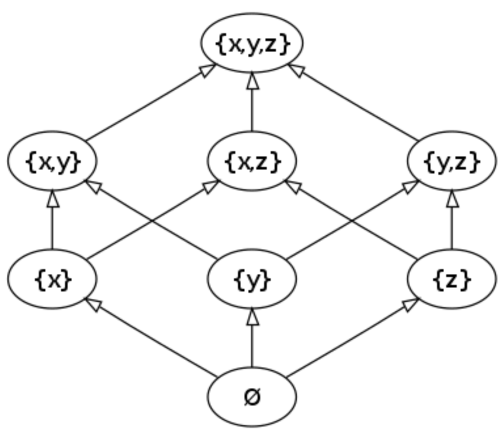


Chain

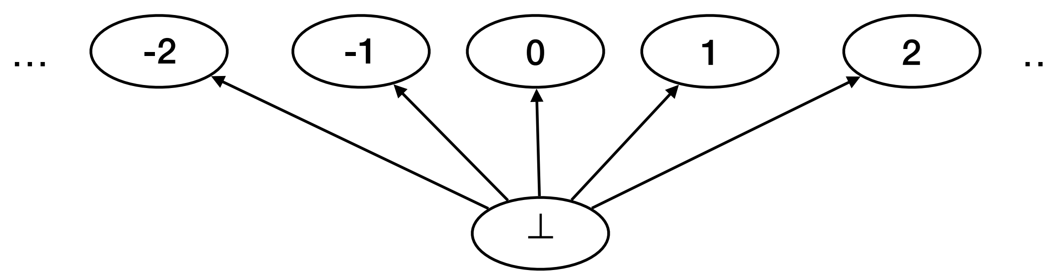
Definition (Chain). Let $(D, \sqsubseteq)$ be a partial ordered set. A subset $X \subseteq D$ is called **chain** if X is totally ordered:

$$\forall x_1, x_2 \in X.\ x_1 \sqsubseteq x_2 \text{ or } x_2 \sqsubseteq x_1.$$

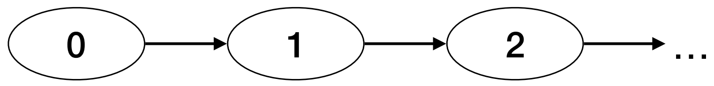
- Example 1: $(\wp(\{x, y, z\}), \subseteq)$



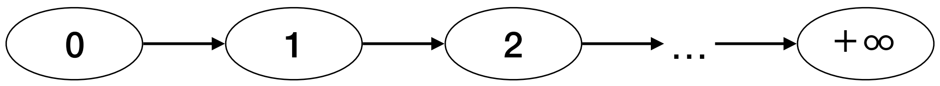
- Example 2: $(\mathbb{Z}_\bot, \sqsubseteq)$



- Example 3: $(\mathbb{N}, \leq)$



- Example 4: $(\mathbb{N} + \{+\infty\}, \leq)$



CPO

Definition (CPO). A poset $(D, \sqsubseteq)$ is a **CPO** (complete partial order) if every chain X of D has $\bigsqcup X \in D$.

Lemma. If poset $(D, \sqsubseteq)$ is a CPO, it has the **least element** $\bot = \bigsqcup \emptyset$

Monotone and Continuous Functions

Definition (Monotone Function). Given two partially ordered sets D_1 and D_2, a function $f : D_1 \to D_2$ is **monotone** if it preserves orders between any two elements in D_1

$$\forall x, y \in D_1.\ x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$$

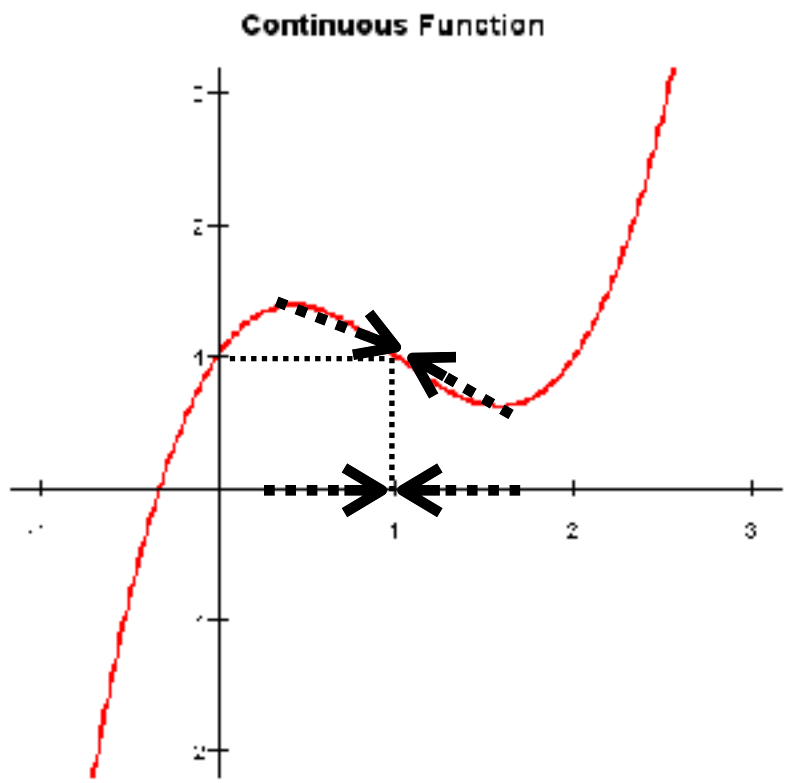
Intuition: the more accurate the input, the more accurate the output

Definition (Continuous Function). Given two partially ordered sets D_1 and D_2, a function $f : D_1 \to D_2$ is **continuous** if it preserves least upper bounds of chains:

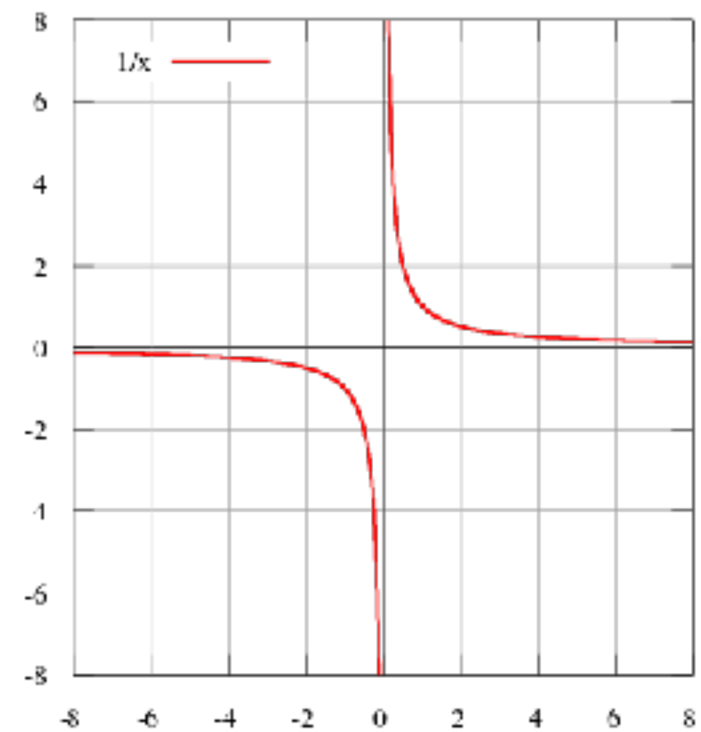
$$\forall chain\ X \subseteq D_1.\ \bigsqcup_{x \in X} f(x) = f(\bigsqcup X).$$

Intuition: the function of the limit is the same as the limit of the functions

Continuous Functions



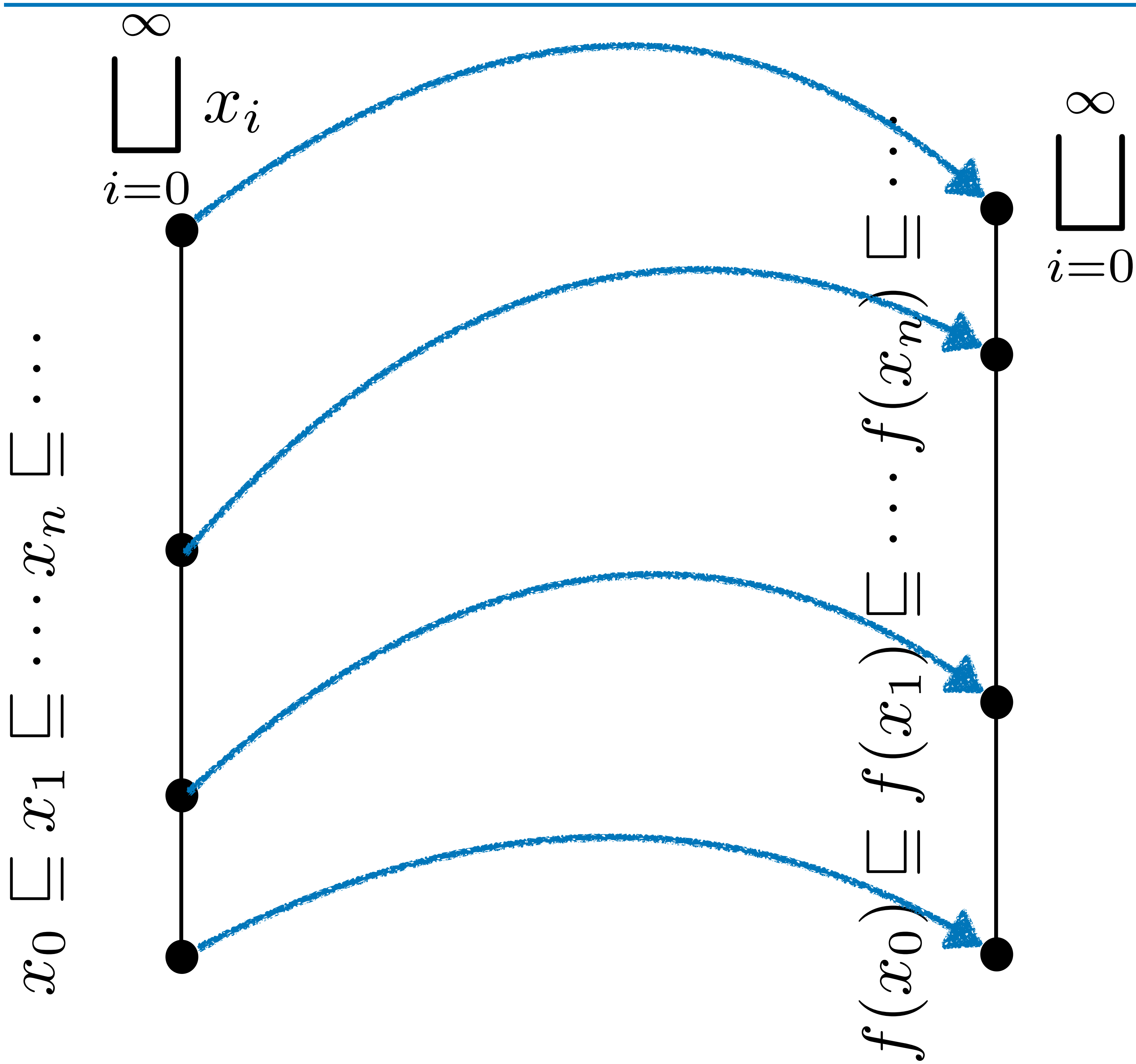
Continuous



Not continuous

$$\forall c.\ \lim_{x \to c} f(x) = f(\lim_{x \to c} x)$$

Analogy

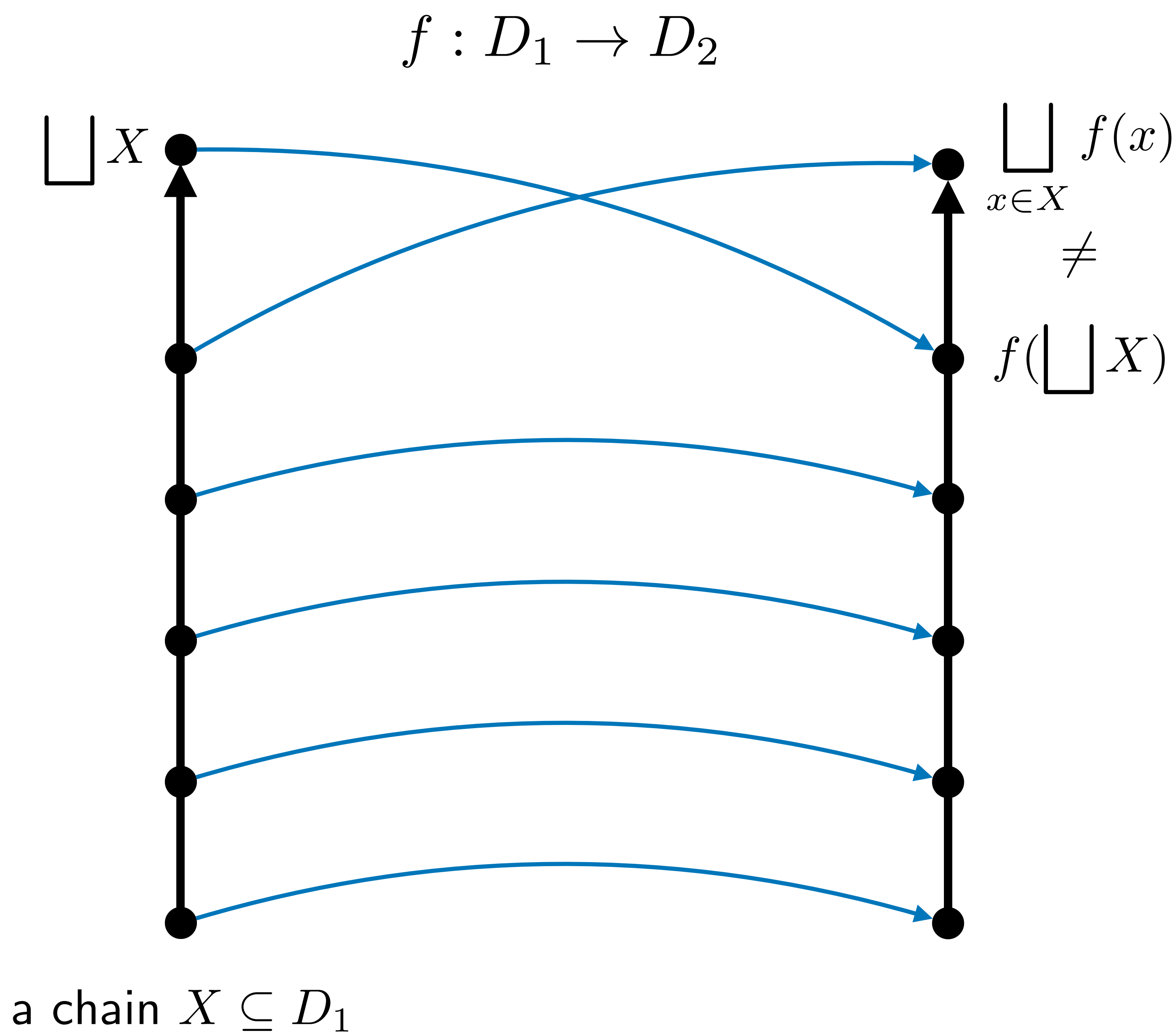


$$\lim_{i \to \infty} f(x_i) = f(\lim_{i \to \infty} x_i)$$

$$\cong$$

$$\bigsqcup_{i=0}^{\infty} f(x_i) = f(\bigsqcup_{i=0}^{\infty} x_i)$$

Non-continuous Function



Properties of Continuous Functions

Lemma 1. *If a function f is continuous, f is monotone.*

Proof. We will show that for any elements a and b such that $a \sqsubseteq b$, $f(a) \sqsubseteq f(b)$.

$$\begin{array}{rcll} f(b) & = & f(a \sqcup b) & (\because a \sqsubseteq b) \\ & = & f(a) \sqcup f(b) & (\text{by continuity of } f) \\ & \sqsupseteq & f(a) & (\text{by definition of } \sqcup) \end{array}$$

□

Properties of Continuous Functions — Fixed points

Definition (Fixed Point). Let $(D, \sqsubseteq)$ be a partial ordered set. A **fixed point** of a function $f : D \to D$ is an element x such that $f(x) = x$. We write $\mathbf{lfp} f$ for the **least fixed point** of f such that

$$f(\mathbf{lfp} f) = \mathbf{lfp} f \quad \text{and} \quad \forall d \in D.\ f(d) = d \implies \mathbf{lfp} f \sqsubseteq d$$

Theorem (Kleene Fixed Point). Let $f : D \to D$ be a continuous function on a CPO D. Then f has the **least fixed point** $\mathbf{lfp} f$ and

$$\mathbf{lfp} f = \bigsqcup_{i \geq 0} f^i(\bot)$$

$$\bot \sqcup f(\bot) \sqcup f(f(\bot)) \sqcup \cdots$$

Proof

$$\mathbf{lfp} f = \bigsqcup_{i \geq 0} f^i(\bot)$$

- Plans: It is enough to show the following two things:

  (1) There exists the chain $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \cdots$ and its least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ in D

  (2) The least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ is the least fixed point of f

Proof

(1) There exists the chain $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \cdots$ and its least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ in D

Proof. We show by induction that $\forall n \in \mathbb{N}.\ f^n(\bot) \sqsubseteq f^{n+1}(\bot)$:

- $\bot \sqsubseteq f(\bot)$ ($\bot$ is the least element of the CPO)
- $f^n(\bot) \sqsubseteq f^{n+1}(\bot) \implies f^{n+1}(\bot) \sqsubseteq f^{n+2}(\bot)$ (by monotonicity of f)

By definition of CPO, least upper bounds of all chains are also in the CPO. Therefore, the least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ of the above chain is in D.

Proof

(2) The least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ is the least fixed point of f

The proof consists of two parts:

(2-1) $\bigsqcup_{i \geq 0} f^i(\bot)$ is a fixed point of f

(2-2) $\bigsqcup_{i \geq 0} f^i(\bot)$ is smaller than all the other fixed points

Proof

(2-1) $\bigsqcup_{i\geq 0} f^i(\bot)$ is a fixed point of f

Proof.

$$
\begin{aligned}
f(\bigsqcup_{n\geq 0} f^n(\bot)) &= \bigsqcup_{n\geq 0} f(f^n(\bot)) && \text{(by continuity of } f\text{)} \\
&= \bigsqcup_{n\geq 0} f^{n+1}(\bot) \\
&= \bigsqcup_{n\geq 0} f^n(\bot)
\end{aligned}
$$

Proof

(2-2) $\bigsqcup_{i \geq 0} f^i(\bot)$ is smaller than all the other fixed points

Proof. Suppose d is a fixed point, i.e., $d = f(d)$. We show that any element $f^i(\bot)$ is smaller than d by induction:

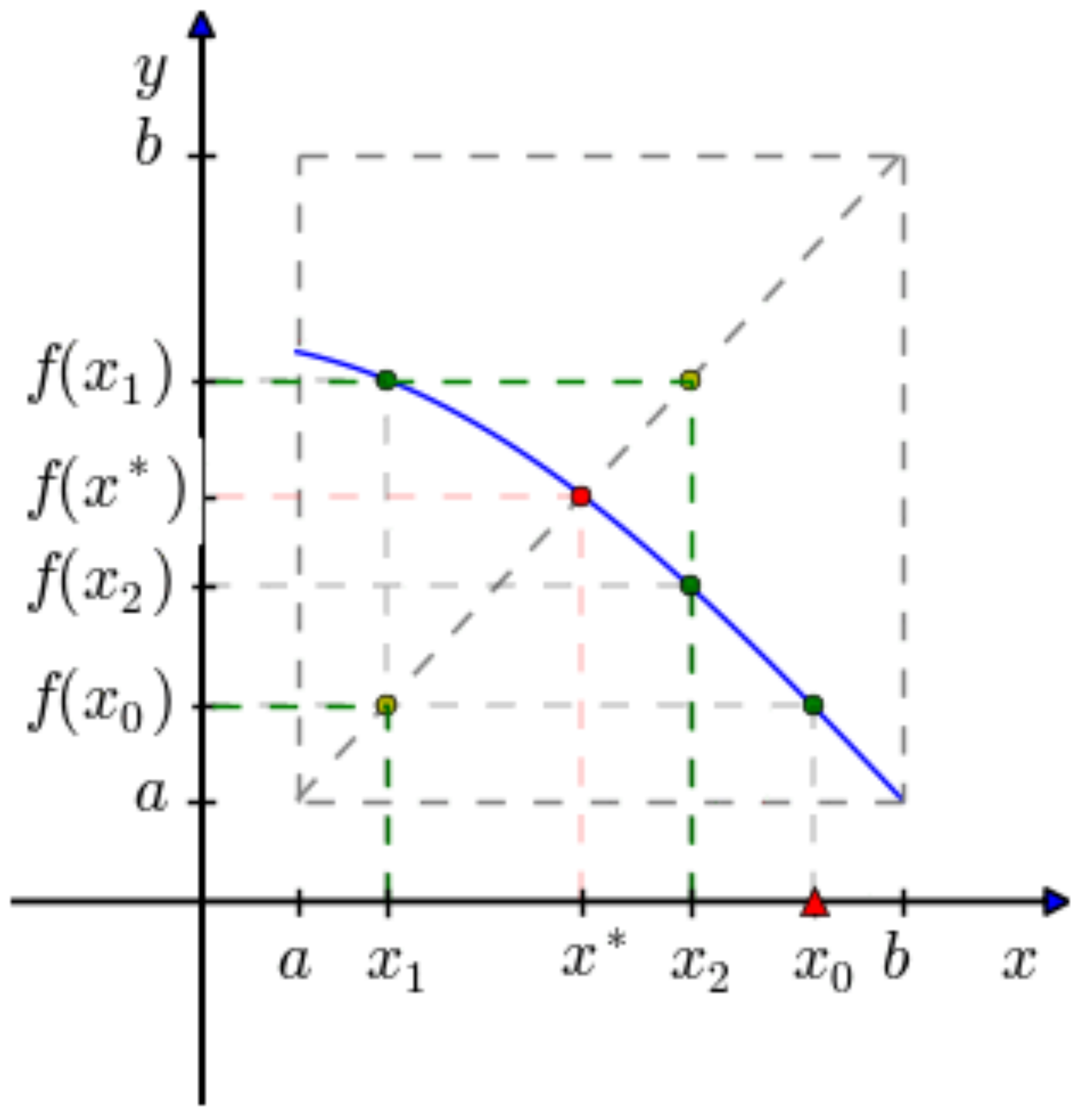
$$\forall n \in \mathbb{N}.\ f^n(\bot) \sqsubseteq d.$$

- $\bot \sqsubseteq d$ ($\bot$ is the least element of the CPO)
- $f^n(\bot) \sqsubseteq d \implies f^{n+1}(\bot) \sqsubseteq f(d) = d$ (by monotonicity of f)

Because all the elements $f^i(\bot)$ are smaller than d, their least upper bound $\bigsqcup_{i \geq 0} f^i(\bot)$ is also smaller than d. Therefore

$$\bigsqcup_{i \geq 0} f^i(\bot) = \mathbf{lfp} f$$

Analogy



https://en.wikipedia.org/wiki/Fixed-point_iteration

Example (While)

- `while (x < 10) x := x + 1`

$$[\![\texttt{while}\ (\texttt{x} < 10)\ \texttt{x} := \texttt{x} + 1]\!] = \lambda m. \begin{cases} [\![\texttt{while}\ (\texttt{x} < 10)\ \texttt{x} := \texttt{x} + 1]\!]([\![x := x + 1]\!](m)) & \text{if } [\![\texttt{x} < 10]\!](m) = \texttt{true} \\ m & \text{if } [\![\texttt{x} < 10]\!](m) = \texttt{false} \end{cases}$$

$$[\![\texttt{while}\ (\texttt{x} < 10)\ \texttt{x} := \texttt{x} + 1]\!] = \mathbf{lfp}\mathcal{F} \text{ where } \mathcal{F}(X) = \lambda m. \begin{cases} X([\![x := x + 1]\!](m)) & \text{if } [\![\texttt{x} < 10]\!](m) = \texttt{true} \\ m & \text{if } [\![\texttt{x} < 10]\!](m) = \texttt{false} \end{cases}$$

$$\mathbf{lfp}\mathcal{F} = \bot \sqcup \mathcal{F}(\bot) \sqcup \mathcal{F}^2(\bot) \sqcup \cdots$$

Example (While)

$$\mathcal{F}(X) = \lambda m. \begin{cases} X(\llbracket x := x+1 \rrbracket(m)) & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{true} \\ m & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{false} \end{cases}$$

$$\bot$$

0 iter

$$\mathcal{F}(\bot) = \lambda m. \begin{cases} \bot(\llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket(m)) & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{true} \\ m & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{false} \end{cases}$$

0, 1 iter

$$\mathcal{F}^2(\bot) = \lambda m. \begin{cases} \mathcal{F}(\bot)(\llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket(m)) & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{true} \\ m & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{false} \end{cases}$$

$$= \lambda m. \begin{cases} \begin{cases} \bot(\llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket^2(m)) & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(\llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket(m)) = \mathtt{true} \\ \llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket(m) & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(\llbracket \mathtt{x} := \mathtt{x}+1 \rrbracket(m)) = \mathtt{false} \end{cases} & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{true} \\ m & \text{if } \llbracket \mathtt{x} < 10 \rrbracket(m) = \mathtt{false} \end{cases}$$

0,1,2 iters

$$\mathcal{F}^3(\bot) = \cdots$$

Constructions of CPOs

- If S is a set, and D_1 and D_2 are CPOs, then the followings are CPOs
  - Lifted set : $D = S_\perp$
  - Cartesian product : $D = D_1 \times D_2$
  - Separated sum : $D = D_1 + D_2$
  - Function : $D = D_1 \to D_2$

Lifted CPO

- $D = S_\perp$

> For any set S, let $D = S + \{\perp\}$ where $\perp$ is an element not in S. Then $(D, \sqsubseteq)$ is a CPO where
>
> $$d \sqsubseteq d' \iff (d = d') \vee (d = \perp)$$

- Why CPO?

Cartesian product

- $D = D_1 \times D_2$

Given two CPOs $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$, $(D, \sqsubseteq)$ is a CPO where

$$D = D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \ \wedge \ d_2 \in D_2\}$$

$$(d_1, d_2) \sqsubseteq (d_1', d_2') \iff (d_1 \sqsubseteq_1 d_1') \wedge (d_2 \sqsubseteq_2 d_2')$$

- Why CPO?

Separated Sum

$D = D_1 + D_2$

Given two CPOs $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$, $(D, \sqsubseteq)$ is a CPO where

$$D = D_1 + D_2 = \{(d_1, 1) \mid d_1 \in D_1\} \cup \{(d_2, 2) \mid d_2 \in D_2\} \cup \{\bot\}$$

$$(d_1, 1) \sqsubseteq (d_1', 1) \iff d_1 \sqsubseteq_1 d_1'$$

$$(d_2, 2) \sqsubseteq (d_2', 2) \iff d_2 \sqsubseteq_2 d_2'$$

- Why CPO?

Function

$D = D_1 \to D_2$

Given two CPOs $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$, $(D, \sqsubseteq)$ is a CPO where

$$D = D_1 \to D_2 = \{f \mid f : D_1 \to D_2 \text{ is a continuous function}\}$$

$$f \sqsubseteq f' \iff \forall d_1 \in D_1.\ f(d_1) \sqsubseteq_2 f'(d_1)$$

- Why CPO?

Proof. Let say we have a chain in D which is $f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \cdots$. We will show that the least upper bound $\bigsqcup_{i \geq 0} f_i$ is in D.

$$\begin{array}{ll} \forall x \in D_1.\ f_0(x) \sqsubseteq_2 f_1(x) \sqsubseteq_2 f_2(x) \sqsubseteq_2 \cdots & \text{(by definition of } \sqsubseteq) \\ \forall i.\ f_i(x) \sqsubseteq_2 \bigsqcup_{i \geq 0} f_i(x) & \text{(by definition of lub)} \end{array}$$

We define $\bigsqcup_{i \geq 0} f_i$ to be $\lambda x.\ \bigsqcup_{i \geq 0} f_i(x)$. Here, $\bigsqcup_{i \geq 0} f_i(x)$ is in D_2 because D_2 is a CPO. Therefore, $\bigsqcup_{i \geq 0} f_i$ is an element of D. $\square$

Summary

- Language = syntax + semantics
- Syntax and semantics are inductively defined.
- Structural induction is a technique for proving interesting properties of inductively defined sets.
- Denotational semantics describes mathematical meaning of programs
  - Semantics is the least fix point of a continuous function