Domain Theory

Recursive Definitions

\[ f(n) = \begin{cases} 1 & \text{if } n = 0 \\ f(n-1) & \text{else} \end{cases} \]

or

\[ \text{define } f = \lambda n . (\text{if } (\text{zerop } n) 1 (f (\text{sub } n 1))) \]

\[ \text{define } g = \lambda n . (\text{if } (\text{zerop } n) 1 (g (\text{succ } n))) \]

A function satisfies a recursive definition iff it is a solution to an equation:

\[ f = \lambda n . (\text{if } (\text{zerop } n) 1 (f (\text{sub } n 1))) \]

\[ g = \lambda n . (\text{if } (\text{zerop } n) 1 (g (\text{succ } n))) \]

Similar to solving a mathematical equation:

\[ x = x^2 - 4x + 6. \]

Other Recursive Definitions

Concrete Syntax

\[ <\text{cmd}> ::= \text{if } <\text{boolean expr}> \text{ then } <\text{cmd} \text{ seq}> \text{ end if} \]

\[ <\text{cmd} \text{ seq}> ::= <\text{cmd}> \mid <\text{cmd}> ; <\text{cmd} \text{ seq}> \]

Lists of Numbers

\[ \text{List} = \{\text{nil}\} \cup (\text{N} \times \text{List}) \]

where nil represents the empty list

Model for Pure Lambda Calculus

\[ V = \text{set of variables} \]

\[ D = V \cup (D \rightarrow D) \]

Problem with Cardinality

\[ |D\rightarrow D| \leq |D| \leq |P(D)| \leq |D\rightarrow D| \]

Modeling Nontermination

Domains

Sets with a lattice-like structure.

Each domain contains a bottom element \( \bot \) that is “less than” all other elements.

For domains of functions, bottom represents a computation that fails to complete normally.

Partial Order \( \subseteq \) on a Set \( S \)

A relation that is

- reflexive
- transitive
- antisymmetric

Definitions

\( b \in S \) is a lower bound of a subset \( A \) of \( S \) if \( b \leq x \) for all \( x \in A \).

\( u \in S \) is an upper bound of a subset \( A \) of \( S \) if \( x \leq u \) for all \( x \in A \).

A least upper bound of \( A \), \( \text{lub } A \), is an upper bound of \( A \) that is less than or equal every upper bound of \( A \).

Example: Divides relation on \{ 1,2,4,5,8,10,20 \}

\[
\begin{align*}
lub \{ 2, 5 \} &= 10 \\
lub \{ 2, 4, 5, 10 \} &= 20 \\
lub \{ 1, 2, 4 \} &= 8 \\
lub \{ 8, 10 \} &= 20 \\
lub \{ 20 \} &= 0
\end{align*}
\]
An *ascending chain* in a partially ordered set $S$ is a sequence of elements $\{x_1, x_2, x_3, x_4, \ldots \}$ with the property
$$x_1 \leq x_2 \leq x_3 \leq x_4 \leq \ldots .$$

A *complete partial order (cpo)* on a set $S$ is a partial order $\leq$ with the two properties
a) There is an element $\bot \in S$ with $\bot \leq x$ for all $x \in S$.
b) Every ascending chain in $S$ has a least upper bound in $S$.

On domains, $\subseteq$ is thought of as *approximates* or is *less defined than or equal to*.

Any finite set with a partial order and a bottom element $\bot$ is a cpo. Why?

---

**Product Domains**

If $A$ with ordering $\subseteq_A$ and $B$ with ordering $\subseteq_B$ are complete partial orders, the *product domain* of $A$ and $B$ is $A \times B$ with the ordering $\subseteq_{A \times B}$ where
$$A \times B = \{<a,b> \mid a \in A \text{ and } b \in B\},$$
and
$$<a,b> \subseteq_{A \times B} <c,d> \iff a \subseteq_A c \text{ and } b \subseteq_B d.$$  

**Thm:** $\subseteq_{A \times B}$ is a partial order on $A \times B$.

**Proof:** Exercise

**Thm:** $\subseteq_{A \times B}$ is a complete partial order on $A \times B$.

**Proof:** $\bot_{A \times B} = <\bot_A, \bot_B>$ acts as bottom for $A \times B$, since $\bot_A \subseteq_A a$ and $\bot_B \subseteq_B b$ for $a \in A$ and $b \in B$.

If $<a_1,b_1> \subseteq <a_2,b_2> \subseteq <a_3,b_3> \subseteq \ldots$ is an ascending chain in $A \times B$,
then $a_1 \subseteq_A a_2 \subseteq_A a_3 \subseteq \ldots$ is a chain in $A$ with least upper bound, $\text{lub} \{a_i \mid i \geq 1\} \in A$, and $b_1 \subseteq_B b_2 \subseteq_B b_3 \subseteq \ldots$ is a chain in $B$ with least upper bound, $\text{lub} \{b_i \mid i \geq 1\} \in B$.

Therefore, $<\text{lub} \{a_i \mid i \geq 1\}, \text{lub} \{b_i \mid i \geq 1\}> \in A \times B$ is the least upper bound for original chain.

---

**Elementary Domains**

Natural numbers and Boolean values with a *discrete partial order*:
$$\text{for } x,y \in S, \quad x \leq y \text{ iff } x = y \text{ or } x = \bot.$$

Elementary domains correspond to “answers”, the results produced by programs:

$\begin{array}{c}
\text{true} \\
\text{false} \\
\bot \\
0 \\
1 \\
2 \\
3 \\
4 \\
\ldots
\end{array}$

Proper and Improper values.

Also called *flat domains*.

---

**Example**

Level = $\{\bot_L, \text{undergraduate}, \text{graduate}, \text{nondegree}\}$

and

Gender = $\{\bot_G, \text{female}, \text{male}\}$

**Gender x Level**

$<f,u> \quad <f,g> \quad <f,n>$

Imagine two processes to determine level and gender of a student.
**Projection Functions**

**first :** $A \times B \rightarrow A$

defined by $\text{first} <a,b> = a$ for any $<a,b> \in A \times B$

and

**second :** $A \times B \rightarrow B$

defined by $\text{second} <a,b> = b$ for any $<a,b> \in A \times B$

Generalize to arbitrary product domains:

$D_1 \times D_2 \times \ldots \times D_n$

**Application:** Calculator Semantics

evaluate $\langle a, \text{op}, d, m \rangle = (a, \text{nop}, \text{op}(a,d), m)$

is a more readable translation of

evaluate $\text{st} =
\langle \text{first}(\text{st}), \text{nop},
\text{second}(\text{st})(\text{first}(\text{st}),\text{third}(\text{st})), \text{fourth}(\text{st})\rangle$.

**Functions on Sum Domains**

Let $S = A + B$.

1. **Injection (creation):**

   $\text{inS : } A \rightarrow S$ is defined for $a \in A$

   as $\text{inS} a = <a,1> \in S$

   $\text{inS : } B \rightarrow S$ is defined for $b \in B$

   as $\text{inS} b = <b,2> \in S$

2. **Projection (selection):**

   $\text{outA : } S \rightarrow A$ is defined for $s \in S$

   $\text{outA} s = a \in A$ if $s=<a,1>$, and

   $\text{outA} s = \bot_A \in A$ if $s=<b,2>$ or $s=\bot_S$.

   $\text{outB : } S \rightarrow B$ is defined for $s \in S$

   $\text{outB} s = b \in B$ if $s=<b,2>$, and

   $\text{outB} s = \bot_B \in B$ if $s=<a,1>$ or $s=\bot_S$.

**Sum Domains**

If $A$ with ordering $\subseteq A$ and $B$ with ordering $\subseteq B$
are complete partial orders, the **sum domain** of $A$ and $B$ is $A + B$ with the ordering $\subseteq_{A+B}$

where

$A + B = \{<a,1> \mid a \in A\} \cup \{<b,2> \mid b \in B\} \cup \{\bot_{A+B}\}$,

$\bot_{A+B} \subseteq A + B <c,1>$ if $a \subseteq A c$,

$\bot_{A+B} \subseteq A + B <d,2>$ if $b \subseteq B d$,

$\bot_{A+B} \subseteq A + B <a,1>$ for each $a \in A$,

$\bot_{A+B} \subseteq A + B <b,2>$ for each $b \in B$, and

$\bot_{A+B} \subseteq A + B \bot_{A+B}$.

**Thm:** $\subseteq_{A+B}$ is a complete partial order on $A + B$.

Proof: $\bot_{A+B} \subseteq x$ for any $x \in A + B$ by definition.

An ascending chain $x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots$ in $A + B$ may repeat $\bot_{A+B}$ forever or eventually climb into either $Ax(1)$ or $Bx(2)$. In first case, least upper bound will be $\bot_{A+B}$, and in the other two cases the least upper bound will exist in $A$ or $B$. $\blacksquare$

3. **Inspection (testing):**

Recall $T = \{\text{true}, \text{false}, \bot_T\}$.

$\text{isA : } S \rightarrow T$ is defined for $s \in S$

$\text{isA} s$ if there exists $a \in A$ with $s=<a,1>$.

$\text{isB : } S \rightarrow T$ is defined for $s \in S$

$\text{isB} s$ if there exists $b \in B$ with $s=<b,2>$.

In both cases, $\bot_S$ is mapped to $\bot_T$.

**Signature Diagram**

Observe that $\text{inS}$ is an overloaded function.
### Storable Values for Wren

- `-1 0 1 2 ...`
- `true false`

#### Functions for SV

- `inSV : Integer → SV`
- `inSV : Boolean → SV`
- `outInteger : SV → Integer`
- `isInteger : SV → T`
- `outBoolean : SV → Boolean`
- `isBoolean : SV → T`

### Functions on D^*

**Injection:**

\[ inD^* : D^k → D^* \]

**Projection:**

\[ outD^k : D^* → D^k \]

#### Functions on Lists:

Let `L ∈ D^*` and `e ∈ D`. Then `L = <d,k>` for `d ∈ D^k` for some `k ≥ 0`

1. **head**: `D^* → D` where
   - `head(L) = first(outD^k(L))` if `k > 0`, and
   - `head(<nil,0>) = ⊥`.

2. **tail**: `D^* → D^*` where
   - `tail(L) = inD^*(<2nd(outD^k(L)), 3rd(outD^k(L)), ..., kth(outD^k(L))>)` if `k > 0`, and
   - `tail(<nil,0>) = ⊥`.

### Generalizations

Finite sums: `D_1 + D_2 + D_3 + ... + D_n`

Infinite sums: `D_1 + D_2 + D_3 + ... = \{<d,i> | d ∈ D_i}\`

Domain of finite Sequences:

\[ D^* = \{nil\} + D + D^2 + D^3 + D^4 + ... \]

where `nil` represents the empty sequence.

### Inspection handled by pattern matching

```plaintext
execute [if E then C] sto =
if p then execute [C] sto
else sto
where bool(p) = evaluate [E] sto,
stands for
execute [if E then C] sto =
if isBoolean(val)
  then if outBoolean(val)
    then execute [C] sto
    else sto
  else ⊥
where val = evaluate [E] sto
```

### Domain of finite Sequences:

\[ D^* = \{nil\} + D + D^2 + D^3 + D^4 + ... \]

where `nil` represents the empty sequence.

\[ SV = \text{Integer} + \text{Boolean} = \{<n,1> | n ∈ \text{Integer}\} ∪ \{<b,2> | b ∈ \text{Boolean}\} ∪ \{⊥_SV\} \]

or

\[ SV = \text{int}(\text{Integer}) + \text{bool}(\text{Boolean}) = \{\text{int}(n) | n ∈ \text{Integer}\} ∪ \{\text{bool}(b) | b ∈ \text{Boolean}\} ∪ \{⊥_SV\} \]

#### Injection function

\[ inSV : \text{Integer} → SV \]

where `inSV n = int(n)`.

The tags (constructors), `int` and `bool`, take the place of the overloaded injection function, `inSV`.

- `int : Integer → SV`
- `bool : Boolean → SV`

#### Projection function

- `outInteger : SV → Integer`
- `isInteger : SV → T`
- `outBoolean : SV → Boolean`
- `isBoolean : SV → T`

- `outD^0 : D^0 → T` if `k > 0`, and
- `outD^0 (nil) = ⊥`.

- `inD^k : D^k → D^*`

- `first(outD^k(L))` if `k > 0`, and
- `(nil,0) = ⊥`.

- `kth(outD^k(L))` if `k > 0`, and
- `(nil,0) = ⊥`.

1. **head**: `D^* → D` where
   - `head(L) = first(outD^k(L))` if `k > 0`, and
   - `head(<nil,0>) = ⊥`.

2. **tail**: `D^* → D^*` where
   - `tail(L) = inD^*(<2nd(outD^k(L)), 3rd(outD^k(L)), ..., kth(outD^k(L))>)` if `k > 0`, and
   - `(nil,0) = ⊥`.
null: D* → T where
null (<nil,0>) = true, and
null (L) = false if L = <d,k> with k > 0.

Therefore, null (L) = isD^2(L)

prefix: DxD* → D* where
prefix (e,L) = inD* (<e, 1st (outD^k(L)),
2nd (outD^k(L)), ..., kth (outD^k(L)))>

affix: D*D → D* where
affix (L,e) = inD* (<1st (outD^k(L)),
2nd (outD^k(L)), ..., kth (outD^k(L)), e>)

Each of these five functions map bottom to bottom.
The binary functions prefix and affix
produce ⊥ if either argument is bottom.

Sets of Functions

A function from a set A to a set B is total if f(x) ∈ B is defined for every x ∈ A.

If A with ordering ⊆A and B with ordering ⊆B
are complete partial orders, define Fun(A,B) to be the set of all total functions from A to B.

Define ⊆ on Fun(A,B) as follows:
For f,g ∈ Fun(A,B),
f ⊆ g if f(x) ⊆B g(x) for all x ∈ A.

Lemma: ⊆ is a partial order on Fun(A,B).
Proof: Follow the definition to show reflexive, transitive, and antisymmetric.
See text for complete proof.

Restrictions on Fun(A,B)

Monotonic
A function f in Fun(A,B) is monotonic if x ⊆A y implies f(x) ⊆B f(y) for all x,y ∈ A.

If ⊆ means "approximates", then when y has at least as much information as x, it follows that f(y) has at least as much information as f(x).

Continuous
A function f ∈ Fun(A,B) is continuous if it preserves least upper bounds; that is, if X = x_1 ⊆A x_2 ⊆A x_3 ⊆A ... is an ascending chain in A, then f(lub_A X) = lub_B {f(x) | x ∈ X}.
Also written f(lub_A {x_i}) = lub_B {f(x_i)}
or f|(lub_A {x_i | i ≥ 1}) = lub_B {f(x_i) | i ≥ 1}.
No surprises when taking the least upper bounds (limits) of approximations.
Lemma: If \( f \in \text{Fun}(A, B) \) is continuous, then it is monotonic.

Proof: Suppose \( f \) is continuous and \( x \subseteq A \ y \). Then \( x \subseteq A \ y \subseteq A \ y \subseteq A \) ... is an ascending chain in \( A \), and since \( f \) is continuous,

\[
f(x) \subseteq B \ lub_B(f(x), f(y)) = f\left(lub_A(x, y)\right) = f(y).
\]

Function Domains

Define \( A \to B \) to be the set of functions in \( \text{Fun}(A, B) \) that are (monotonic and) continuous. This set is ordered by the relation \( \subseteq \) from \( \text{Fun}(A, B) \).

\[
F \in \text{Fun}(N \to N, N \to N) \text{ defined by } \quad F \ g = \lambda \ n \ . \ \text{if } g(n) = \perp \text{ then } 0 \text{ else } 1, \quad \text{for } g \in N \to N
\]
is not monotonic.

Proof by Counterexample:
Let \( g_1 = \lambda \ n \ . \ \perp \text{ and } g_2 = \lambda \ n \ . \ 0 \). Then \( g_1 \subseteq G_2 \).
But \( F(g_1) = \lambda \ n \ . \ 0 \), \( F(g_2) = \lambda \ n \ . \ 1 \), and functions \( \lambda n \ . \ 0 \text{ and } \lambda n \ . \ 1 \) are not related at all by \( \subseteq \).

Thm: The relation \( \subseteq \) on \( A \to B \) is a complete partial order.

Proof: Since \( \subseteq \) is a partial order on \( A \to B \), two properties need to be verified:
1. The bottom element in \( \text{Fun}(A, B) \) is also in \( A \to B \); that is, the function \( \perp(x) = \perp_B \) is monotonic and continuous.

2. For any ascending chain in \( A \to B \), its least upper bound, which is an element of \( \text{Fun}(A, B) \), is also in \( A \to B \), namely it is monotonic and continuous.

Part 1: If \( x \subseteq A \ y \) for some \( x, y \in A \), then \( \perp(x) = \perp_B = \perp_B \), which means \( \perp(x) \subseteq B \perp(y) \), and so \( \perp \) is a monotonic function.

If \( x_1 \subseteq A \ x_2 \subseteq A \ x_3 \subseteq A \) ... is an ascending chain in \( A \), then its image under the function \( \perp \) will be the ascending chain \( \perp_B \subseteq B \perp_B \subseteq B \perp_B \subseteq B \perp_B \subseteq B \ldots \), whose least upper bound is \( \perp_B \). Therefore, \( \perp\left(lub_A\{x_i|\ i \geq 1\}\right) = \perp_B = lub_B(\perp(x_i)|\ i \geq 1\) \), and \( \perp \) is a continuous function.

Lemma: The relation \( \subseteq \) restricted to \( A \to B \) is a partial order.

Proof: The properties reflexive, transitive, and antisymmetric are inherited by a subset.

Lub Lemma: If \( x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots \) is an ascending chain in a cpo \( A \), and \( x_i \subseteq d \) for each \( i \geq 1 \), then \( lub \{x_i|\ i \geq 1\} \subseteq d \).

Proof: By the definition of least upper bound, if \( d \) is a bound for the chain, the least upper bound, \( lub \{x_i|\ i \geq 1\} \), must be no larger than \( d \).

Limit Lemma: If \( x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots \) are ascending chains in cpo \( A \), and \( x_i \subseteq y_i \) for each \( i \geq 1 \), then \( lub \{x_i|\ i \geq 1\} \subseteq lub \{y_i|\ i \geq 1\} \).

Proof: For each \( i \geq 1 \), \( x_i \subseteq y_i \subseteq lub \{y_i|\ i \geq 1\} \).
Therefore \( lub \{x_i|\ i \geq 1\} \subseteq lub \{y_i|\ i \geq 1\} \) by the Lub lemma (take \( d = lub \{y_i|\ i \geq 1\} \)).

Part 2: Let \( f_1 \subseteq f_2 \subseteq f_3 \subseteq \ldots \) be an ascending chain in \( A \to B \), and let \( F = lub \{f_i|\ i \geq 1\} \) be its least upper bound (in \( \text{Fun}(A, B) \)). Remember the definition of \( F \), \( F(x) = lub \{f_i(x)|\ i \geq 1\} \) for each \( x \in A \).

We need to show that \( F \) is monotonic and continuous so that we know \( F \) is a member of \( A \to B \).

Monotonic:
If \( x \subseteq A \ y \), \( f_i(x) \subseteq B \ f_i(y) \subseteq B lub \{f_i(y)|\ i \geq 1\} \) for any \( i \) since each \( f_i \) is monotonic.

Therefore, \( F(y) = lub \{f_i(y)|\ i \geq 1\} \) is an upper bound for each \( f_i(x) \), and so the least upper bound of all the \( f_i(x) \) satisfies \( F(x) = lub \{f_i(x)|\ i \geq 1\} \subseteq F(y) \) (Lub lemma), and \( F \) is monotonic.

Continuous: Let \( x_1 \subseteq A \ x_2 \subseteq A \ x_3 \subseteq A \) ... be an ascending chain in \( A \). We need to show that \( F(lub \{x_i|\ i \geq 1\}) = lub \{F(x_i)|\ i \geq 1\} \) where \( F(x) = lub \{f_i(x)|\ i \geq 1\} \) for each \( x \in A \).
Note that “i” is used to index the ascending chain of functions from A→B while “j” is used to index the ascending chains of elements in A and B.

So F is continuous if

\[ F(\text{lub } \{x_j|j \geq 1\}) = \text{lub } \{f_i(x_j)|i \geq 1\} \]

for each chain \( \{x_j|j \geq 1\} \) in A.

Recall these definitions and properties:

1. Each \( f_i \) is continuous:
   \[ f_i(\text{lub } \{x_j|j \geq 1\}) = \text{lub } \{f_i(x_j)|j \geq 1\} \]

   for each chain \( \{x_j|j \geq 1\} \) in A.

2. Definition of F:
   \[ F(x) = \text{lub } \{f_i(x)|i \geq 1\} \]

   for each \( x \in A \).

So

\[ \text{lub } \{f_i(x)|i \geq 1\} = \text{lub } \{f_i(\text{lub } \{x_j|j \geq 1\})|i \geq 1\} \]

by 2

\[ = \text{lub } \{f_i(x_j)|i \geq 1\} \]

by 1

\[ = \text{lub } \{f_i(x_j)|i \geq 1\} \]

by \( \forall \)

\[ = \text{lub } \{F(x_j)|j \geq 1\} \]

by 2.

Look at Figure 10.9.

**Second Half**

\[ \text{lub } \{\text{lub } \{f_i(x_j)|i \geq 1\}|j \geq 1\} \subseteq \text{lub } \{\text{lub } \{f_i(x_j)|i \geq 1\}|i \geq 1\} \]

For all i and k,

\[ f_i(x_k) \subseteq \text{lub } \{f_i(x_j)|j \geq 1\} \]

by using the fact that each \( f_i \) is monotonic and continuous (the columns of Figure 10.9).

We have chains

\[ f_1(x_k) \subseteq f_2(x_k) \subseteq f_3(x_k) \subseteq \ldots \]

for each k

\[ \text{lub } \{f_1(x_j)|j \geq 1\} \subseteq \text{lub } \{f_2(x_j)|j \geq 1\} \subseteq \text{lub } \{f_3(x_j)|j \geq 1\} \subseteq \ldots \]

So for each k,

\[ \text{lub } \{f_i(x_k)|i \geq 1\} \subseteq \text{lub } \{\text{lub } \{f_i(x_j)|i \geq 1\}|j \geq 1\} \]

by the Limit lemma.

This corresponds to the rightmost column.

Hence

\[ \text{lub } \{\text{lub } \{f_i(x_k)|i \geq 1\}|k \geq 1\} \subseteq \text{lub } \{\text{lub } \{f_i(x_j)|i \geq 1\}|i \geq 1\} \]

by the Lub lemma. Now change k to j.

Therefore F is continuous.

**Example 10**

**Student = \{ \bot, Autry, Bates \}**

**Level = \{ \bot, undergraduate, graduate, nondegree \}**

Fun(Student, Level) contains 64 (4^3) elements. Only 19 of these functions are monotonic and continuous.

Which of these functions are monotonic?

\[ f = \{ \bot \mapsto \bot, \text{ Autry } \mapsto \text{nondegree}, \text{ Bates } \mapsto \bot \} \]

\[ g = \{ \bot \mapsto \text{grad}, \text{ Autry } \mapsto \text{grad}, \text{ Bates } \mapsto \bot \} \]

\[ h = \{ \bot \mapsto \text{grad}, \text{ Autry } \mapsto \text{grad}, \text{ Bates } \mapsto \text{grad} \} \]
Chapter 10

3. An ascending chain \( s_1 \subseteq s_2 \subseteq s_3 \subseteq \ldots \) in \( S = A + B \) may repeat \( \#A+\#B \) forever or eventually climb into \( A \). Since \( A \) is finite, for some \( k \), \( x_k = x_{k+1} = x_{k+2} = \ldots \).

So the chain is a finite set, \( \{x_1, x_2, x_3, \ldots, x_k\} \), whose least upper bound is \( x_k \).

Since \( f \) is monotonic,
\[
f(x_1) \subseteq_B f(x_2) \subseteq_B f(x_3) \subseteq_B \ldots \subseteq_B f(x_k)
\]
is an ascending chain in \( B \), which is also a finite set, namely \( \{f(x_1), f(x_2), f(x_3), \ldots, f(x_k)\} \).

Therefore, \( f(\text{lub}\{x_i|i=1\}) = f(x_k) = \text{lub}\{f(x_i)|i=1\} \), and \( f \) is continuous.

Thm: If \( A \) and \( B \) are cpos, \( A \) is a finite set, and \( f \in \text{Fun}(A, B) \) is monotonic, \( f \) is also continuous.

Proof: Let \( x_1 \subseteq A \), \( x_2 \subseteq A \), \( x_3 \subseteq A \ldots \) be an ascending chain in \( A \).

Since \( A \) is finite, for some \( k \), \( x_k = x_{k+1} = x_{k+2} = \ldots \).

So the chain is a finite set, \( \{x_1, x_2, x_3, \ldots, x_k\} \), whose least upper bound is \( x_k \).

Since \( f \) is monotonic,
\[
f(x_1) \subseteq_B f(x_2) \subseteq_B f(x_3) \subseteq_B \ldots \subseteq_B f(x_k)
\]
is an ascending chain in \( B \), which is also a finite set, namely \( \{f(x_1), f(x_2), f(x_3), \ldots, f(x_k)\} \).

Therefore, \( f(\text{lub}\{x_i|i=1\}) = f(x_k) = \text{lub}\{f(x_i)|i=1\} \), and \( f \) is continuous.

Continuity of Functions on Domains

Thm: These functions on domains and their analogs are continuous:

1. \( \text{first} : A \times B \to A \)
2. \( \text{in}_S : A \to S \) where \( S = A + B \)
3. \( \text{out}_A : A + B \to A \)
4. \( \text{is}_A : A + B \to \top \)

Case 1:
For some \( k \geq 1 \), \( s_i = <a_i,2> \) for all \( i \geq k \), where \( a_i \in A \).

Then \( \text{out}_A(\text{lub}\{s_i|i\geq1\}) = \text{out}_A(\top_S) = \top_A \), and \( \text{lub}\{\text{out}_A(s_i)|i\geq1\} = \text{lub}\{\top_A|i\geq1\} = \top_A \).

Case 2:
For some \( k \geq 1 \), \( s_i = <a_i,1> \) for all \( i \geq k \), where \( a_i \in A \).

Then \( \text{out}_A(\text{lub}\{s_i|i\geq1\}) = \text{out}_A(<\text{lub}\{a_i|i\geq1\},1>) = \text{lub}\{a_i|i\geq1\} \)

and \( \text{lub}\{\text{out}_A(s_i)|i\geq1\} = \text{lub}\{a_i|i\geq1\} \).

Case 3:
For some \( k \geq 1 \), \( s_i = <b_i,2> \) for all \( i \neq k \), where \( b_i \in B \).

Then \( \text{out}_A(\text{lub}\{s_i|i\geq1\}) = \text{out}_A(<\text{lub}\{b_i|i\geq1\},2>) = \top_A \)

and \( \text{lub}\{\text{out}_A(s_i)|i\geq1\} = \text{lub}\{\top_A|i\geq1\} = \top_A \).

Thm: The composition of continuous functions is continuous.

Proof: Suppose \( f : A \to B \) and \( g : B \to C \) are continuous functions.

Let \( x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots \) be an ascending chain in \( A \).

Then \( f(x_1) \subseteq f(x_2) \subseteq f(x_3) \subseteq \ldots \) is an ascending chain in \( B \) with \( \text{out}_B(\text{lub}\{x_i|i\geq1\}) = \text{lub}(f(x_i)|i\geq1) \) by the continuity of \( f \).

Since \( g \) is continuous,
\[
g(f(x_1)) \subseteq g(f(x_2)) \subseteq g(f(x_3)) \subseteq \ldots \text{ is an ascending chain in } C \text{ with } g(\text{lub}(f(x_i)|i\geq1)) = \text{lub}(g(f(x_i))|i\geq1).
\]

Therefore \( g(\text{lub}(f(x_i)|i\geq1)) = g(\text{lub}(f(x_i))|i\geq1) = \text{lub}(g(f(x_i))|i\geq1) \) and \( g \circ f \) is continuous.

Chapter 10
Fixed Point Semantics

**Goal:** Provide meaning for recursive definitions.

**First Step:** Transform partial functions into total functions.

**Example**

f is a function with domain D = {0,1,2} and codomain C = {0,1,2} defined by:

\[ f(n) = \frac{2n}{n} \]

or

\[ f = \{<1,2>,<2,1>,<0,?>\}. \]

Note that f(0) is undefined; therefore f is a partial function.

Now extend f to make it a total function:

\[ f = \{<1,2>,<2,1>,<0,?>\}. \]

**Thm:** Let \( f^+ \) be a natural extension of a function between two sets D and C so that \( f^+ \) is a total function from \( D^+ \) to \( C^+ \).

Then \( f^+ \) is monotonic and continuous.

Proof: Let \( x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots \) be an ascending chain in the domain \( D^+ = D \cup \{\bot\} \).

Two possibilities for the behavior of the chain:

**Case 1:** \( x_i = \bot_D \) for all \( i \geq 1 \).

Then \( \text{lub}(x_i | i \geq 1) = \bot_D \), and

\[ f^+ (\text{lub}(x_i | i \geq 1)) = f^+(\bot_D) = \bot_C = \text{lub}(\bot_C) = \text{lub}(f^+(x_i | i \geq 1)). \]

**Case 2:** \( x_i = \bot_D \) for \( 1 \leq i \leq k \) and \( x_{k+1} = x_{k+2} = x_{k+3} = \ldots \), since once the terms move above bottom, the sequence is constant in a flat domain.

Then \( \text{lub}(x_i | i \geq 1) = x_{k+1} \), and

\[ f^+ (\text{lub}(x_i | i \geq 1)) = f^+(x_{k+1}) = \text{lub}(\bot_C, f^+(x_{k+1})) = \text{lub}(f^+(x_i | i \geq 1)). \]

If \( f^+ \) is continuous, it is also monotonic.

---

Add an undefined element to the codomain, \( C^+ = \{\bot, 0, 1, 2\} \), and for symmetry, do likewise with the domain, \( D^+ = \{\bot, 0, 1, 2\} \).

Define the **natural extension** of f by having \( \bot_D \) map to \( \bot_C \) under f:

\[ f^+ = \{<\bot,\bot>, <0,\bot>, <1,2>, <2,1>\}. \]

Define a relationship that orders functions and domains according to how “defined” they are, putting a lattice-like structure on the elementary domains:

For \( x, y \in D^+ \), \( x \subseteq y \) if \( x = \bot \) or \( x = y \).

This relation is read “x approximates y” or “x is less defined or equal to y.”

---

The **natural extension** of a function whose domain is a Cartesian product, namely f : \( D_1^+ \times D_2^+ \times \ldots \times D_n^+ \rightarrow C^+ \), has the property that \( f^+(x_1, x_2, \ldots, x_n) = \text{lub} \) whenever at least one \( x_i = \bot \).

Any function that satisfies this property is known as a **strict** function.

**Thm:** If \( f^+: D_1^+ \times D_2^+ \times \ldots \times D_n^+ \rightarrow C^+ \) is a natural extension where \( D_i^+ \), 1 ≤ i ≤ n, and \( C^+ \) are elementary domains, then \( f^+ \) is monotonic and continuous.

Proof: Consider the case where \( n = 2 \). Show \( f^+ \) is continuous.

Let \( <x_1, y_1> \subseteq <x_2, y_2> \subseteq <x_3, y_3> \subseteq \ldots \) be an ascending chain in \( D_1^+ \times D_2^+ \). Since \( D_1^+ \) and \( D_2^+ \) are elementary domains, the chains \( \{x_i | i \geq 1\} \) and \( \{y_i | i \geq 1\} \) must follow one of the two cases in the previous proof, namely all \( \bot \) or eventually a constant proper value in \( D_1^+ \).
Case 1: \( \text{lub}(x_i| i \geq 1) = \bot_{D_1^+} \) or \( \text{lub}(y_i| i \geq 1) = \bot_{D_2^+} \) (or both).
Then \( f^*(\text{lub}(x_i, y_i| i \geq 1)) = f^*(\text{lub}(x_i| i \geq 1), \text{lub}(y_i| i \geq 1)) = \bot_{C^+} \)
because \( f^* \) is a natural extension and one of its arguments is \( \bot \), and
\[
\text{lub}(f^*(<x_i, y_i>| i \geq 1)) = \text{lub}(\bot_{C^+}, f^*(<x_i, y_i>))
\]
and \( \text{lub}(f^*(<x_i, y_i>| i \geq 1)) = \text{lub}(\bot_{C^+}, \text{lub}(x_i, y_i)) = f^*(<x_i, y_i>). \)

Case 2: \( \text{lub}(x_i| i \geq 1) = x \in D_1 \) and \( \text{lub}(y_i| i \geq 1) = y \in D_2 \)
Since \( D_1^+ \) and \( D_2^+ \) are both elementary domains, there is an integer \( k \)
such that \( x_i = x \) and \( y_i = y \) for all \( i \leq k \).
So \( f^*(\text{lub}(x_i, y_i| i \geq 1)) = f^*(<x_i, y_i>) \in C^+ \)
and \( \text{lub}(f^*(<x_i, y_i>)| i \geq 1) = \text{lub}(\bot_{C^+}, f^*(<x_i, y_i>)) = f^*(<x_i, y_i>). \)

Example

Consider the natural extension of the conditional expression operation:
\[
(\text{if } a \text{ } b \text{ } c) = \text{if } a \text{ then } b \text{ else } c.
\]
The natural extension unduly restricts the meaning of the conditional expression.
For example, we prefer that the following expression return 0 when \( x = 1 \) and \( y = 0 \):
\[
\text{if } y > 0 \text{ then } x/y \text{ else } 0.
\]
If we interpret the undefined operation \( 1/0 \) as \( \bot \),
when \( x = 1 \) and \( y = 0 \),
\[
(\text{if}^+ \ y > 0 \ x/y \ 0) = (\text{if}^+ \ \text{false} \ 1) = 1
\]
for a natural extension.

Second Step: Give meaning to recursive definitions.

Consider a recursively defined function \( f : N \rightarrow N \) where \( N = \{ \bot, 0, 1, 2, 3, \ldots \} \) and
\[
f(n) = \begin{cases} 
5 & \text{if } n = 0 \\
\text{if } n = 1 \text{ then } f(n+2) \text{ else } f(n-2) & \text{if } n \neq 0 \text{ and } n \neq 1
\end{cases}
\]

Two questions:
1. What function, if any, does this equation in \( f \) denote?
2. Does it specify more than one function?

Define a functional \( F \) by
\[
F : (N \rightarrow N) \rightarrow (N \rightarrow N) \text{ where}
\]
\[
(F(f))(n) = \begin{cases} 
5 & \text{if } n = 0 \\
\text{if } n = 1 \text{ then } f(n+2) \text{ else } f(n-2) & \text{if } n \neq 0 \text{ and } n \neq 1
\end{cases}
\]

Function application associates to the left; omit the parentheses with multiple applications, writing \( F \ f \ n \) for \( (F(f))(n) \).
**Fixed Points in Mathematics**

<table>
<thead>
<tr>
<th>Function</th>
<th>Fixed Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(n) = n^2 - 6n )</td>
<td>0 and 7</td>
</tr>
<tr>
<td>( g(n) = n )</td>
<td>all ( n \in \mathbb{N} )</td>
</tr>
<tr>
<td>( g(n) = n + 5 )</td>
<td>none</td>
</tr>
<tr>
<td>( g(n) = 2 )</td>
<td>2</td>
</tr>
</tbody>
</table>

**Back to Functional F**

The function \( g = \lambda n \cdot 5 \) is a fixed point of \( F \):

\[
F \ g = \lambda n \cdot 5 \quad \text{if} \ n=0 \ \text{then 5 else if} \ n=1 \ \text{then} \ g(n+2) \quad \text{else} \ g(n-2) = \lambda n \cdot 5 \quad \text{if} \ n=0 \ \text{then 5 else if} \ n=1 \ \text{then} \ 5 \ \text{else 5} = \lambda n \cdot 5 = g.
\]

**Problem**

\( g = \lambda n \cdot 5 \) does not agree with the operational behavior of the original recursive definition.

\[
f(1) = f(3) = f(1) = \ldots \text{does not produce a value, whereas} \ g(1) = 5.
\]

**Thm:** If \( D \) with \( \subseteq \) is a complete partial order and \( g : D \rightarrow D \) (\( g \) is any monotonic and continuous function on \( D \)), then \( g \) has a least fixed point with respect to \( \subseteq \) on \( D \rightarrow D \).

**Proof:** Since \( D \) is a cpo, \( g^0(\bot) = \bot \subseteq g(\bot) \).

Since \( g \) is monotonic, \( g(\bot) \subseteq g(g(\bot)) = g^2(\bot) \).

In general, since \( g \) is monotonic,

\[
g^i(\bot) \subseteq g^{i+1}(\bot) \quad \text{implies} \quad g^{i+1}(\bot) = g(g^i(\bot)) \subseteq g^{i+1}(\bot) = g^{i+2}(\bot).
\]

So by induction,

\[
\bot \subseteq g(\bot) \subseteq g^2(\bot) \subseteq g^3(\bot) \subseteq g^4(\bot) \subseteq \ldots
\]

is an ascending chain in \( D \), which must have a least upper bound \( u = \text{lub}(g^i(\bot) \mid i \geq 0) \in D \).

But \( g(u) = g(\text{lub}(g^i(\bot) \mid i \geq 0)) = \text{lub}(g(g(\bot)) \mid \bot \subseteq g(\bot)) \) since \( g \) is continuous

\[
= \text{lub}(g^{i+1}(\bot) \mid i \geq 0) = \text{lub}(g^i(\bot) \mid i \geq 0) = u
\]

That is, \( u \) is a fixed point for \( g \).

Note that \( g^0(\bot) = \bot \) has no effect on the least upper bound of \( \{g^j(\bot) \mid j \geq 0\} \).

Let \( \nu \in D \) be another fixed point for \( g \).

Then \( \bot \subseteq \nu \) and \( g(\bot) \subseteq g(\nu) = \nu \), the basis step for induction.

Suppose \( g^i(\bot) \subseteq \nu \).

Then since \( g \) is monotonic,

\[
g^{i+1}(\bot) = g(g^i(\bot)) \subseteq g(\nu) = \nu, \text{ the induction step.}
\]

Therefore, by mathematical induction, \( g(\bot) \subseteq \nu \) for all \( i \geq 0 \).

So \( \nu \) is an upper bound for \( \{g^j(\bot) \mid j \geq 0\} \).

Hence \( u \subseteq \nu \), since \( u \) is the least upper bound for \( \{g^j(\bot) \mid j \geq 0\} \).

**Corollary:** Every continuous functional \( F : (A \rightarrow B) \rightarrow (A \rightarrow B) \), where \( A \) and \( B \) are domains, has a least fixed point, \( F_p : A \rightarrow B \), which can be taken as the meaning of the (recursive) definition corresponding to \( F \).
Example
Consider the functional G: (N→N)→(N→N) where
\[ G \, g \, n \, = \, \begin{cases} 1 & \text{if } n = 0 \text{ then } \frac{g(3)-12}{4n+g(n-2)} \quad (\dagger) \\ 1 & \text{else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(n-2)} \quad (\ddagger) \\ a & \text{else} \end{cases} \]
that corresponds to the recursive definition
\[ g(n) = \begin{cases} 1 & \text{if } n = 0 \text{ then } 1 \\ 9 & \text{else if } n = 1 \text{ then } 9 \\ 25 & \text{else if } n = 2 \text{ then } 25 \\ \vdots & \text{else} \end{cases} \]
Contemplate the ascending sequence
\[ \frac{\downarrow}{\downarrow} \subseteq \frac{G(\downarrow)}{\downarrow} \subseteq \frac{G^2(\downarrow)}{\downarrow} \subseteq \frac{G^3(\downarrow)}{\downarrow} \subseteq \ldots \]
and its least upper bound.
Use the abbreviation \( g_k = (G^k \downarrow) \) for \( k \geq 0 \):
\[ g_0(n) = G^0 \downarrow n = \downarrow(n) \]
\[ g_1(n) = G \downarrow n = G g_0 n \]
\[ g_2(n) = G (G \downarrow) n = G g_1 n \]
\[ g_3(n) = G^3 \downarrow n = G g_2 n \]
Now calculate a few terms in the ascending chain
\[ g_0 \subseteq g_1 \subseteq g_2 \subseteq g_3 \subseteq \ldots \]

Note Property
\[ a + (\text{if } b \text{ then } c \text{ else } d) = \begin{cases} a & \text{if } b \text{ then } a+c \text{ else } a+d \\ \end{cases} \]
\[ g_3(n) = G^3 \downarrow n = G g_2 n \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(n-2)} \\ 9 & \text{else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(n-2)} \end{cases} \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } \downarrow \text{ else if } n = 1 \text{ then } \downarrow \\ 9 & \text{else if } n = 2 \text{ then } \downarrow \\ 25 & \text{else if } n = 3 \text{ then } \downarrow \text{ else if } n = 4 \text{ then } \downarrow \text{ else if } n = 5 \text{ then } \downarrow \text{ else if } n = 6 \text{ then } 49 \text{ else } \downarrow \end{cases} \]
\[ g_4(n) = G^4 \downarrow n = G g_3 n \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(3)(n-2)} \\ 9 & \text{else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(3)(n-2)} \end{cases} \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } \downarrow \text{ else if } n = 1 \text{ then } \downarrow \\ 9 & \text{else if } n = 2 \text{ then } \downarrow \\ 25 & \text{else if } n = 3 \text{ then } \downarrow \text{ else if } n = 4 \text{ then } 25 \text{ else } \downarrow \text{ else if } n = 5 \text{ then } \downarrow \\ 49 & \text{else if } n = 6 \text{ then } 49 \text{ else } \downarrow \end{cases} \]

\[ g_0(n) = G^0 \downarrow n = \downarrow(n) = \downarrow \text{ for } n \in \mathbb{N}, \]
the everywhere undefined function.
\[ g_1(n) = G \downarrow n = G g_0 n \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(3)(n-2)} \\ 9 & \text{else if } n = 1 \text{ then } \frac{g(3)-12}{4n+g(3)(n-2)} \end{cases} \]
\[ = \begin{cases} 1 & \text{if } n = 0 \text{ then } \downarrow \text{ else if } n = 1 \text{ then } \downarrow \\ 9 & \text{else if } n = 2 \text{ then } \downarrow \\ 25 & \text{else if } n = 3 \text{ then } \downarrow \text{ else if } n = 4 \text{ then } 25 \text{ else } \downarrow \text{ else if } n = 5 \text{ then } \downarrow \\ 49 & \text{else if } n = 6 \text{ then } 49 \text{ else } \downarrow \end{cases} \]
A pattern seems to be developing.

**Lemma**: For all $i \geq 0$,
\[
g_i(n) = \begin{cases} 
  & \text{if } n < 2i \text{ then } (\text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot) \text{ else } \bot, \\
  & \text{if } n < 2i \text{ then } (\text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot) \text{ else } \bot, \\
  & \text{for any arbitrary } i \geq 0.
\end{cases}
\]

Proof: The proof proceeds by induction on $i$.

a) By the previous computations, for $i = 0$,
\[
go_0(n) = \bot = \begin{cases} 
  & \text{if } n < 2 \text{ then } (\text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot) \text{ else } \bot.
\end{cases}
\]

b) As the induction hypothesis, assume that
\[
g_i(n) = \begin{cases} 
  & \text{if } n < 2i \text{ then } (\text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot) \text{ else } \bot, \\
  & \text{for any arbitrary } i \geq 0.
\end{cases}
\]

Then $g_{i+1}(n) = G g_i n$
\[
= \begin{cases} 
  & \text{if } n = 0 \text{ then } 1 \\
  & \text{else if } n = 1 \text{ then } g_i(3)-12 \text{ else } 4n+g_i(n-2)
\end{cases}
\]

Therefore our pattern for the $g_i$ is correct. □

The least upper bound of the ascending chain $g_0 \subseteq g_1 \subseteq g_2 \subseteq g_3 \subseteq \ldots$, where $g_i(n) = \begin{cases} 
  & \text{if } n < 2i \text{ then } (\text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot) \text{ else } \bot, \\
  & \text{for any arbitrary } i \geq 0.
\end{cases}$

must be defined (not $\bot$) for any $n$ where some $g_i$ is defined, and must take the value $(n+1)^2$ there.

Hence the least upper bound is
\[
G_{fp}(n) = (\text{lub}(g_i \mid i \geq 0)) n
= (\text{lub}(G^1 \bot \mid i \geq 0)) n
= \text{if } \text{even}(n) \text{ then } (n+1)^2 \text{ else } \bot
\]
for all $n \in \mathbb{N}$, and this function can be taken as the meaning of the original recursive definition.

Note that the function $h n = (n+1)^2$ is also a fixed point for $G$.

It is more defined than $G_{fp}$.

In fact, $G_{fp} \subseteq h$.

**fix**

The procedure for computing a least fixed point for a functional can be described as an operator on functions $F : D \rightarrow D$:
\[
\text{fix} : (D \rightarrow D) \rightarrow D
\]
where
\[
\text{fix} F = \text{lub}(F^i(\bot) \mid i \geq 0) \in D.
\]

The least fixed point of the functional $F = \lambda f . \lambda n . \text{if } n = 0 \text{ then } 5 \text{ else } \text{if } n = 1 \text{ then } f(n+2) \text{ else } f(n-2)$ can then be expressed as
\[
F_{fp} = \text{fix } F, \text{ an element of } D = \mathbb{N} \rightarrow \mathbb{N}.
\]

For $F : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$, fix has type
\[
\text{fix} : ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}).
\]

The fixed point operator $\text{fix}$ provides a fixed point for any continuous functional, namely, the least defined function with this fixed point property.

**Fixed Point Identity**: $F(\text{fix } F) = \text{fix } F$. 

\[\begin{align*}
\text{if } n = 0 \text{ then } & 1 \\
\text{else if } n = 1 \text{ then } & -12 \\
\text{else } & 4n + \text{if } n < 2i \text{ then } \text{if even}(n-2) \text{ then } (n-1)^2 \text{ else } \bot \text{ else } \bot \\
\text{then } & \text{if even}(n-2) \text{ then } (n-1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } 4n + (n-1)^2 \text{ else } 4n + \bot \text{ else } 4n + \bot \\
\text{then } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{then } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\text{else } & \text{if even}(n-2) \text{ then } (n+1)^2 \text{ else } \bot \\
\end{align*}\]
Chapter 10

Continuous Functionals

**Lemma:** A constant function \( f : D \to C \), where \( f(x) = k \) for some fixed \( k \in C \) and for all \( x \in D \), is continuous given either of the two extensions:

a) The natural extension where \( f(\bot_D) = \bot_C \).
b) The “unnatural” extension where \( f(\bot_D) = k \).

**Lemma:** An identity function \( f : D \to D \), where \( f(x) = x \) for all \( x \) in a domain \( D \), is continuous.

Proof: If \( x_1 \subseteq x_2 \subseteq x_3 \subseteq ... \) is an ascending chain in \( D \), it follows that

\[
f(\text{lub}\{x_i|i \leq n\}) = \text{lub}\{x_i|i \leq n\} = \text{lub}\{f(x_i)|i \leq n\}.
\]

**Conditional Expression Function:**

Natural extension of “if” is too restrictive.

**Lazy if**

\[
\text{if}(a,b,c) = \text{if } a \text{ then } b \text{ else } c.
\]

where \( \text{if} : T \times D \times D \to D \) for some domain \( D \)

and \( T = \{ \bot, \text{true}, \text{false} \} \)

Chapter 10

Composition Involving a Parameter

\( F : (N \to N) \to (N \to N) \) where \( F f n = n + \text{if } n = 0 \text{ then } 0 \text{ else } f(f(n-1)) \).

Let \( f_1 \leq f_2 \leq f_3 \leq ... \) be a chain in \( D \).

The proof shows that \( \text{lub}\{F f_i|i \leq n\} = F(\text{lub}\{f_i|i \leq n\}) \)

in two parts.

Chapter 10

Part 1: \( \text{lub}\{F(f_i)|i \leq n\} \subseteq F(\text{lub}\{f_i|i \leq n\}) \).

For each \( i \geq 1 \), \( f_i \subseteq \text{lub}\{f_i|i \leq n\} \).

Since \( F_1 \) is monotonic, \( F_1(f_i) \subseteq F_1(\text{lub}\{f_i|i \leq n\}) \),

which means that

\( F_1 f_i d \subseteq F_1 \text{lub}\{f_i|i \leq n\} d \) for each \( d \in D \).

Since \( f_i \) is monotonic, \( f_i(F_1 f_i d) \subseteq F_1 \text{lub}\{f_i|i \leq n\} d \).

But \( F_1 f_i d = f_i < F_1 f_i d > \) and

\( f_i < F_1 \text{lub}\{f_i|i \leq n\} d > \subseteq \text{lub}\{f_i|i \leq n\} < F_1 \text{lub}\{f_i|i \leq n\} d > \).

Therefore, \( F_1 f_i d \subseteq \text{lub}\{f_i|i \leq n\} < F_1 \text{lub}\{f_i|i \leq n\} d > \) for each \( i \geq 1 \) and \( d \in D \).

So, \( \text{lub}\{F(f_i)|i \leq n\} d = \text{lub}\{F f_i d|i \leq n\} \subseteq \text{lub}\{f_i|i \leq n\} < F_1 \text{lub}\{f_i|i \leq n\} d > = F \text{lub}\{f_i|i \leq n\} d \) for \( d \in D \).

Part 2: \( F(\text{lub}\{f_i|i \leq n\}) \subseteq \text{lub}\{F(f_i)|i \leq n\} \).

For any \( d \in D \), \( F \text{lub}\{f_i|i \leq n\} d = \text{lub}\{f_i|i \leq n\} < F_1 \text{lub}\{f_i|i \leq n\} d > \) by defn of \( F_1 \).

= \text{lub}\{f_i|i \leq n\} \text{lub}\{F(f_i)|i \leq n\} d \) since \( F_1 \) is cont,

= \text{lub}\{f_i|i \leq n\} < F_1 \text{lub}\{f_i|i \leq n\} d > \text{lub}\{f_i|i \leq n\} d \)

since \( \text{lub}\{f_i|i \leq n\} \) is continuous.

= \text{lub}\{\text{lub}\{f_i(i\{F_1(f_i)|i \leq n\})d\}|i \leq n\} \}

by definition of \( \text{lub}\{f_i|i \leq n\} \). \( \dagger \)
Fixed Points for Nonrecursive Functions

Find the least fixed point for the function

\( h(n) = n^3 - 3n \)

as a rule defining a "recursive" function that just has no actual recursive call of \( h \).

The corresponding functional

\( H : (N \to N) \to (N \to N) \) is defined by the rule:

\[ H \ h \ n = n + \text{if } n=0 \text{ then } 0 \text{ else } h(n-1) \]

Second Interpretation:

Think of \( h(n) = n^3 - 3n \) as a rule defining a recursive function that just has no actual recursive call of \( h \).

The fixed point construction:

\[ H^0 \perp n = \perp(n) = \perp \]
\[ H^1 \perp n = n^3 - 3n \]
\[ H^2 \perp n = n^3 - 3n \]
\[ H^3 \perp n = n^3 - 3n \]

Therefore, the least fixed point is

\( \text{lub}(H^k(\perp)) = \perp \) for \( k \geq 0 \), which follows the same definition rule as the original function \( h \).
Revisiting Denotational Semantics

The recursive definition

\[
\text{execute } \llbracket \text{while } E \text{ do } C \rrbracket \text{ sto } =
\]
\[
\text{if } \text{evaluate } \llbracket E \rrbracket \text{ sto } = \text{bool}(\text{true})
\]
\[
\text{then execute } \llbracket \text{while } E \text{ do } C \rrbracket (\text{execute } \llbracket C \rrbracket \text{ sto})
\]
\[
\text{else sto}
\]
violates the principle of compositionality.

The function \( \text{execute } \llbracket \text{while } E \text{ do } C \rrbracket \) satisfies the recursive definition above if and only if it is a fixed point of the functional

\[
W f s = \text{if } \text{evaluate } \llbracket E \rrbracket s = \text{bool}(\text{true})
\]
\[
\text{then } f(\text{execute } \llbracket C \rrbracket s) \text{ else } s
\]
\[
= \text{if } \text{evaluate } \llbracket E \rrbracket s = \text{bool}(\text{true})
\]
\[
\text{then } (f \circ \text{execute } \llbracket C \rrbracket) s \text{ else } s.
\]

We obtain a nonrecursive and compositional definition of the meaning of a \text{while} command by means of

\[
\text{execute } \llbracket \text{while } E \text{ do } C \rrbracket = \text{fix } W.
\]

We gain insight into both the \text{while} command and fixed point semantics by constructing a few terms in the ascending chain whose least upper bound is \text{fix } W,

\[
W^0 \downarrow \subseteq W^1 \downarrow \subseteq W^2 \downarrow \subseteq W^3 \downarrow \subseteq \ldots
\]

where \text{fix } W = \text{lub } \{W^i | i \in \mathbb{N}\}.

The fixed point construction for \( W \):

\[
W^0 \downarrow s = \bot
\]
\[
W^1 \downarrow s = W (W^0 \downarrow) s
\]
\[
= \text{if } \text{evaluate } \llbracket E \rrbracket s = \text{bool}(\text{true})
\]
\[
\text{then } \bot (\text{execute } \llbracket C \rrbracket s) \text{ else } s
\]
\[
= \text{if } \text{evaluate } \llbracket E \rrbracket s = \text{bool}(\text{true})
\]
\[
\text{then } \bot \text{ else } (\text{execute } \llbracket C \rrbracket s)
\]

Let \text{exC} stand for the function \text{execute } \llbracket C \rrbracket.
Chapter 10 67
Fixed Point Induction
Induction on the construction of the least fixed point \( \text{lub} \{ F^i \perp | i \geq 0 \} \).

Let \( \Phi(f) \) be a predicate that describes a property for an arbitrary function \( f \) defined recursively.

To show \( \Phi \) holds for the least fixed point \( F_0 \) of the functional \( F \) corresponding to a recursive definition of \( f \), two conditions are needed:

Part 1: Show by induction that \( \Phi \) holds for each element in the ascending chain
\( \perp \subseteq F \subseteq F^2 \subseteq F^3 \subseteq \ldots \) and

Part 2: Show that \( \Phi \) remains true when the least upper bound is taken.

Part 2 is handled by defining a class of predicates with the necessary property.

A predicate is called admissible if it has the property that whenever the predicate hold for an ascending chain of functions, it also must hold for the least upper bound of that chain.

The least upper bound of this ascending sequence provides semantics for the while command:
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{fix} \ W = \text{lub} \{ W^i \perp | i \geq 0 \}.
\]

View the definition of \( \text{execute} \ [\text{while} \ E \ do \ C] \) in terms of the fixed point identity,
\[
W(f) s = \text{if evaluate} \ [E] s = \text{bool}(\text{true})
\]
then \( f(\text{execute} \ [C] s) \) else \( s \).

In this context,
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{fix} \ W
\]

Now define \( \text{loop} = \text{fix} \ W \). Then
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{loop}
\]
where \( \text{loop} s = (W \ \text{loop}) s \)
= \( \text{loop} \) where \( \text{loop} s = \text{if evaluate} \ [E] s = \text{bool}(\text{true}) 
\text{then execute} \ [C] s \) else \( s \).

This approach produces the compositional definition of \( \text{execute} \ [\text{while} \ E \ do \ C] \) used in the specification of Wren, Figure 9.11.

Chapter 10 66
The least upper bound of this ascending sequence provides semantics for the while command:
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{fix} \ W = \text{lub} \{ W^i \perp | i \geq 0 \}.
\]

View the definition of \( \text{execute} \ [\text{while} \ E \ do \ C] \) in terms of the fixed point identity,
\[
W(f) s = \text{if evaluate} \ [E] s = \text{bool}(\text{true})
\]
then \( f(\text{execute} \ [C] s) \) else \( s \).

In this context,
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{fix} \ W
\]

Now define \( \text{loop} = \text{fix} \ W \). Then
\[
\text{execute} \ [\text{while} \ E \ do \ C] = \text{loop}
\]
where \( \text{loop} s = (W \ \text{loop}) s \)
= \( \text{loop} \) where \( \text{loop} s = \text{if evaluate} \ [E] s = \text{bool}(\text{true}) 
\text{then execute} \ [C] s \) else \( s \).

This approach produces the compositional definition of \( \text{execute} \ [\text{while} \ E \ do \ C] \) used in the specification of Wren, Figure 9.11.

Fixed Point Induction

Induction on the construction of the least fixed point \( \text{lub} \{ F^i \perp | i \geq 0 \} \).

Let \( \Phi(f) \) be a predicate that describes a property for an arbitrary function \( f \) defined recursively.

To show \( \Phi \) holds for the least fixed point \( F_0 \) of the functional \( F \) corresponding to a recursive definition of \( f \), two conditions are needed:

Part 1: Show by induction that \( \Phi \) holds for each element in the ascending chain
\( \perp \subseteq F \subseteq F^2 \subseteq F^3 \subseteq \ldots \) and

Part 2: Show that \( \Phi \) remains true when the least upper bound is taken.

Part 2 is handled by defining a class of predicates with the necessary property.

A predicate is called admissible if it has the property that whenever the predicate hold for an ascending chain of functions, it also must hold for the least upper bound of that chain.

Theorem: Any finite conjunction of inequalities of the form \( \alpha(F) \subseteq \beta(F) \), where \( \alpha \) and \( \beta \) are continuous functionals, is an admissible predicate. This includes terms of the form \( \alpha(F) = \beta(F) \).

Proof: See [Manna72].

Mathematical induction is used to verify the condition in Part 1:

Given a functional \( F : (D \rightarrow D) \rightarrow (D \rightarrow D) \) for some domain \( D \) and admissible predicate \( \Phi(f) \), show:

a) \( \Phi(\perp) \) holds where \( \perp : D \rightarrow D \), and
b) for any \( i \geq 0 \), if \( \Phi(F^i(\perp)) \), then \( \Phi(F^{i+1}(\perp)) \).

An alternate version of condition b) is:

b') for any \( f : D \rightarrow D \), if \( \Phi(f) \), then \( \Phi(F(f)) \).

Either formulation is sufficient to infer that the predicate \( \Phi \) holds for every function in the ascending chain \( \{ F^i \perp | i \geq 0 \} \).
**Example**

H n = if n=0 then 0 else (2n-1)+h(n-1) with least fixed point \(H_\ell\).

Prove that \(H_\ell \subseteq \lambda n . \ n^2\).

Let \(\Phi(f)\) be the predicate \(f \subseteq \lambda n . \ n^2\).

a) Since \(\perp \subseteq \lambda n . \ n^2\), \(\Phi(\perp)\) holds.

b) Suppose \(\Phi(h)\), that is, \(h \subseteq \lambda n . \ n^2\).

Then \(H n = \begin{cases} \text{if } n=0 \text{ then } 0 & \text{else } (2n-1)+h(n-1) \\ \subseteq & \text{if } n=0 \text{ then } 0 & \text{else } (2n-1)+(n-1)^2 \\ = & \text{if } n=0 \text{ then } 0 & \text{else } n^2 \\ = & n^2 \text{ for } n \geq 0. \end{cases}\)

Therefore, \(\Phi(H(h))\) holds, and by fixed point induction \(H_\ell \subseteq \lambda n . \ n^2\).

**Fixed-Point Identity**

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

Add a reduction rule that carries out effect of fixed-point identity from right to left to replicate the functional \(F\)—namely, \(\text{fix } F \Rightarrow F(\text{fix } F)\).

Consider this definition of a function involving powers of 2 with its associated functional:

\[ \text{two } n = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{else } 2 \cdot \text{two}(n-1) + 1 \end{cases} \]

and

\[ \text{Two } = \lambda h . \ \lambda n . \ \text{if } n=0 \text{ then } 1 \text{ else } 2 \cdot h(n-1) + 1. \]

The least fixed point of \(\text{Two}\), \((\text{fix } \text{Two})\), serves as the definition of the two function.

The function \((\text{fix } \text{Two})\) is not recursive and can be “reduced” using the fixed-point identity

\[ \text{fix } \text{Two} \Rightarrow \text{Two } (\text{fix } \text{Two}). \]

**Paradoxical Combinator**

An implementation of the fixed-point operator \(\text{fix}\) in the (untyped) lambda calculus:

\[ \text{define } Y = \lambda f . \ (\lambda x . \ f \ (x \ x)) \ (\lambda x . \ f \ (x \ x)) \]

or in the lambda calculus evaluator

\[ \text{define } Y = (L f ((L x \ (f \ (x \ x))) \ (L x \ (f \ (x \ x))))). \]

Reduction proves \(Y\) satisfies fixed-point identity.

\[ Y \ E = (\lambda f . \ (\lambda x . \ f \ (x \ x)) \ (\lambda x . \ f \ (x \ x))) \ E \]

\[ \Rightarrow (\lambda x . \ E \ (x \ x)) \ (\lambda x . \ E \ (x \ x)) \]

\[ \Rightarrow E \ ((\lambda x . \ E \ (x \ x)) \ (\lambda x . \ E \ (x \ x))) \]

\[ \Rightarrow E \ (\lambda h . \ (\lambda x . \ h \ (x \ x)) \ (\lambda x . \ h \ (x \ x))) \]

\[ \Rightarrow E \ (Y \ E). \]

Calculation follows normal order reduction.

Applicative order strategy leads to a nonterminating reduction:

\[ Y \ E = (\lambda f . \ (\lambda x . \ f \ (x \ x)) \ (\lambda x . \ f \ (x \ x))) \ E \]

\[ \Rightarrow (\lambda f . \ f \ ((\lambda x . \ f \ (x \ x)) \ (\lambda x . \ f \ (x \ x)))) \ E \]

\[ \Rightarrow (\lambda f . \ f \ ((\lambda x . \ f \ (x \ x)) \ (\lambda x . \ f \ (x \ x)))) \ E \]

\[ \Rightarrow \ldots \]

The replication of the function encoded in the \(\text{fix}\) operator enables a reduction to create as many copies of the original function as it needs.

\[ (\text{fix } \text{Two}) \ 4 \]

\[ \Rightarrow (\text{Two } (\text{fix } \text{Two})) \ 4 \]

\[ \Rightarrow (\lambda h . \ \lambda n . \ \text{if } n=0 \text{ then } 1 \text{ else } 2 \cdot h(n-1)+1) \ (\text{fix } \text{Two}) \ 4 \]

\[ \Rightarrow (\lambda n . \ \text{if } n=0 \text{ then } 1 \text{ else } 2 \cdot (\text{fix } \text{Two})(n-1)+1) \ 4 \]

\[ \Rightarrow \text{if } 4=0 \text{ then } 1 \text{ else } 2 \cdot (\text{fix } \text{Two})(4-1)+1 \]

\[ \Rightarrow 2 \cdot ((\text{fix } \text{Two}) \ 3)+1 \]

\[ \Rightarrow 2 \cdot ((\text{Two } (\text{fix } \text{Two})) \ 3)+1 \]

\[ \Rightarrow 2 \cdot ((\lambda h . \ \lambda n . \ \text{if } n=0 \text{ then } 1 \text{ else } 2 \cdot h(n-1)+1) \ (\text{fix } \text{Two}) \ 3)+1 \]
\[ \Rightarrow 2 \cdot ((\lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot (\text{fix } \text{Two}) (n-1)+1) \; 3)+1 \]
\[ \Rightarrow 2 \cdot ((\lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot (\text{fix } \text{Two}) (3-1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot ((\text{fix } \text{Two}) \; 2)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot ((\text{Two} \; (\text{fix } \text{Two})) \; 2)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot ((\lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot (\text{fix } \text{Two}) (n-1)+1) \; 2)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot ((\lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot ((\text{fix } \text{Two}) \; 1)+1)+1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot (2 \cdot (\text{fix } \text{Two}) \; 0)+1)+1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot (2 \cdot ((\lambda h. \; \lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot ((\text{fix } \text{Two}) \; 1)+1)+1)+1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot (2 \cdot ((\lambda n. \; \text{if } n=0 \; \text{then } 1 \; \text{else } 2 \cdot ((\text{fix } \text{Two}) \; (n-1))+1) \; 0)+1)+1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot (2 \cdot ((\text{fix } \text{Two}) \; (0-1))+1)+1)+1)+1)+1 \]
\[ \Rightarrow 2 \cdot (2 \cdot (2 \cdot (2 \cdot 0)+1)+1)+1 = 31 \]