How to define things by recursion

Alex Kavvos

LOGSEM seminar, 11 May 2020

Institut for Datalogi, Aarhus Universitet

Defining functions by recursion

Let

$$\mathrm{fact} : \mathbb{N} \to \mathbb{N}$$

$$\mathrm{fact}(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times \mathrm{fact}(n-1)$$

Is this well-defined?

Operational solution: write an interpreter.

Mathematical solutions:

1. Postulate that "definition by induction" is a thing.
   But what about the following function?

   $$f(n) \stackrel{\text{def}}{=} \textbf{if } n = 1 \textbf{ then } 1 \textbf{ elsif } \mathrm{even}(n) \textbf{ then } f(n/2) \textbf{ else } f(3n+1)$$

2. Write down a little abstract machine. (Implicitly just like 1.)
3. Do a little bit of domain theory. Fun for the whole family!

Embracing higher-order functions

Use λ-abstraction:

$$\text{fact} = \lambda n.\ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times \text{fact}(n-1)$$

A very common form: a function defined in terms of itself.

Abstract the recursive call:

$$\text{fact} = (\lambda f.\lambda n.\ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times f(n-1))\,\text{fact}$$

This is of the form $\text{fact} = F(\text{fact})$, where

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$
$$F(f) = \lambda n.\ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times f(n-1)$$

where $\mathbb{N} \rightharpoonup \mathbb{N}$ is the set of partial *functions* on $\mathbb{N}$.

fact is a **fixed point** of F.

Partiality & Approximation I

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$
$$F(f) = \lambda n.\ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times f(n-1)$$

A curious phenomenon. If $\bot : \mathbb{N} \rightharpoonup \mathbb{N}$ is the undefined function, let

$$f_0 \stackrel{\text{def}}{=} \bot \stackrel{\text{def}}{=} \emptyset \qquad\qquad f_{n+1} \stackrel{\text{def}}{=} F(f_n)$$

Observe that

$$\begin{aligned}
f_1 &= \{(0,1)\} \\
f_2 &= \{(0,1),(1,1)\} \\
f_3 &= \{(0,1),(1,1),(2,2)\} \\
f_4 &= \{(0,1),(1,1),(2,2),(3,6)\} \\
&\vdots
\end{aligned}$$

Partiality & Approximation II

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$
$$F(f) = \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times f(n-1)$$

$$f_0 = \emptyset$$
$$f_1 = \{(0,1)\}$$
$$f_2 = \{(0,1),(1,1)\}$$
$$f_3 = \{(0,1),(1,1),(2,2)\}$$
$$f_4 = \{(0,1),(1,1),(2,2),(3,6)\}$$

Intuitively, fact is the **limit** of this sequence. Some observations:

1. f_{i+1} is **consistent** with f_i.
2. f_{i+1} is **more defined** than f_i.

The Subset Order

1. f_{i+1} is **consistent** with f_i.
2. f_{i+1} is **more defined** than f_i.

Recall the *subset relation* between partial functions:

$$f \subseteq g \quad \stackrel{\text{def}}{\equiv} \quad \forall x, y \in \mathbb{N}.\ (x, y) \in f \Longrightarrow (x, y) \in g$$

g is possibly more defined than f, and agrees with it wherever both are defined. Writing $E \simeq E'$ for *Kleene equality*:

$$f \subseteq g \quad \stackrel{\text{def}}{\equiv} \quad \forall x, y \in \mathbb{N}.\ f(x) \simeq y \implies g(x) \simeq y$$

$\subseteq$ is a relation on $(\mathbb{N} \rightharpoonup \mathbb{N})$. It is a **partial order**:

reflexive $f \subseteq f$

transitive $f \subseteq g \ \wedge \ g \subseteq h \implies f \subseteq h$

antisymmetric $f \subseteq g \ \wedge \ g \subseteq f \implies f = g$

ω-chains

Notice that $F(f) = \lambda n.\ \mathbf{if}\ n = 0\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1)$ is **monotonic**: a more defined input leads to a more defined output.

$$f \subseteq g \implies F(f) \subseteq F(g)$$

We prove by induction that the sequence

$$f_0 \stackrel{\text{def}}{=} \bot \qquad\qquad f_{n+1} \stackrel{\text{def}}{=} F(f_n)$$

is an ω-*chain*:

$$f_0 \subseteq f_1 \subseteq f_2 \subseteq f_3 \subseteq \dots$$

BC: $f_0 \stackrel{\text{def}}{=} \emptyset \subseteq f_1$ whatever f_1 is.

IS: if $f_i \subseteq f_{i+1}$ then $f_{i+1} = F(f_i) \subseteq F(f_{i+1}) = f_{i+2}$ by monotonicity.

Limits

Recap. To define the factorial function:

1. We characterised it as the **fixed point** of
   $F(f) = \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times f(n-1)$.
2. We produced a sequence $(f_i)_{i \in \omega}$ of **approximations** to it.
3. These approximations have a **sense of purpose**: they become progressively more defined, without contradicting previous information.

If we take the set

$$f \stackrel{\text{def}}{=} \bigcup_{i \in \omega} f_i$$

we find that it is a **partial function** itself. (Why?)

$\boxed{f \text{ is the limit of the sequence } (f_i)_{i \in \omega}}$

The Smoking Gun I

It remains to prove that $f \stackrel{\text{def}}{=} \bigcup_{i \in \omega} f_i$ is a fixed point of F.

$$F(f) = F\left(\bigcup_{i \in \omega} f_i\right) = ???$$

Something is missing.

- $f = \bigcup_{i \in \omega} f_i$ is a huge object: it is defined at all natural numbers.
- But $F(f)(n) \stackrel{\text{def}}{=} \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times f(n-1)$ uses the value of f at **a finite number of points**.

F does not make any "decisions" based on the entirety of f.

We say that F is **continuous**.

Continuity

Definition
A monotonic functional $F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ is **continuous** if for any ω-chain $(f_i)_{i \in \omega}$ we know that

$$F\left(\bigcup_{i \in \omega} f_i\right) = \bigcup_{i \in \omega} F(f_i)$$

By monotonicity, we always have $\bigcup_{i \in \omega} F(f_i) \subseteq F\left(\bigcup_{i \in \omega} f_i\right)$. It suffices to check

$$F\left(\bigcup_{i \in \omega} f_i\right) \subseteq \bigcup_{i \in \omega} F(f_i)$$

That is: F cannot make any decisions based on the whole limit!

Example
$F(f) = \textbf{if } (f = \mathrm{id}_{\mathbb{N}}) \textbf{ then } \lambda n.\ 1 \textbf{ else } \lambda n.\ 0$ is not continuous.

The Smoking Gun II

The functional

$$F(f) = \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times f(n-1)$$

is "obviously" continuous: it uses f at a finite number of points.

It remains to prove that $f \stackrel{\text{def}}{=} \bigcup_{i \in \omega} f_i$ is a fixed point of F.

$$F(f) = F\left(\bigcup_{i\in\omega} f_i\right) = \bigcup_{i\in\omega} F(f_i) = \bigcup_{i\in\omega} f_{i+1} = \bigcup_{i\in\omega} f_i$$

(The first term of an ω-chain can be skipped in the union.)

So f is a fixed point.

We may take it as the definition of the factorial function.

This covers all recursive definitions

Let $g : \mathbb{N} \to \mathbb{N}$ be a **total computable function**.

Given the Gödel number $\langle M \rangle$ of a Turing machine M, read $g(\langle M \rangle)$ as the Gödel number $\langle N \rangle$ of another TM N (quite possibly gibberish).

Suppose g is **extensional**: if TMs M and N compute the same function, then so do the TMs encoded by $g(\langle M \rangle)$ and $g(\langle N \rangle)$.

Example
The function that writes out the source code of
$\lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times f(n-1)$ when given the source of f.

This defines a functional

$$F_g : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$$

We call this an **effective operation**.

This covers all recursive definitions

Theorem (Myhill & Sherpherdson, 1955)
Every effective operation $F_g : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ *is*

1. *monotonic*
2. *continuous*
3. *effective on finite functions*

Moreover, every such functional is an effective operation.

The last condition means: there is a program that given the full list of input-output pairs of a finite function $\theta \stackrel{\text{def}}{=} \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and some input x computes $F(\theta)(x)$.

Thus, any reasonable **template**/**specification** has a fixed point. (Reasonable = there is a TM that when given code meant to run at the time of a recursive call outputs code for the entire function definition.)

Abstracting partial functions away

Save the last bit, nothing so far depends on partial functions.

Let $\sqsubseteq$ be a **partial order** on a set D: a reflexive, transitive, antisymmetric relation. The following is akin to a **limit**.

Definition (Least upper bound)
The *least upper bound* of $S \subseteq D$ is an element $\bigsqcup S \in D$ such that

1. $\forall x \in S.\ x \sqsubseteq \bigsqcup S$
2. if $\forall x \in S.\ x \sqsubseteq z$ then $\bigsqcup S \sqsubseteq z$

Example
Let $\mathfrak{W} \subseteq \mathcal{P}(X)$. The least upper bound of $\mathfrak{W}$ in $(\mathcal{P}(X), \subseteq)$ is given by the **union**

$$\bigcup \mathfrak{W} \stackrel{\text{def}}{=} \{x \in X \mid \exists S \in \mathfrak{W}.\ x \in S\}$$

It is the **least** set that contains all the sets in $\mathfrak{W}$.

ω-complete partial orders

Definition (ω-complete partial order)
A partial order $(D, \sqsubseteq)$ is *ω-complete* just if

1. it has a **least element** $\bot$, so that $\forall x \in D.\ \bot \sqsubseteq x$
2. every ω-chain $(x_i)_{i \in \omega}$ has a **least upper bound** $\bigsqcup_{i \in \omega} x_i \in D$.

Let D and E be ω-cpos.

Definition
A function $f : D \to E$ is **monotonic** if $x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$

Definition
A function $f : D \to E$ is **continuous** if for every ω-chain $(x_i)_{i \in \omega}$

$$f\left(\bigsqcup_{i \in \omega} x_i\right) = \bigsqcup_{i \in \omega} f(x_i)$$

The Fixed Point Theorem

Theorem (Kleene ≈1935, Tarski 1939)
Let $f : D \to D$ be a continuous function on an ω-cpo D. Then f has a least fixed point *given by*

$$\mathrm{lfp}(f) \stackrel{\mathit{def}}{=} \bigsqcup_{i\in\omega} f^i(\bot)$$

Proof.
Induction: $\left(f^i(\bot)\right)_{i\in\omega}$ is an ω-chain. The lub is a fixed point:

$$f\left(\bigsqcup_{i\in\omega} f^i(\bot)\right) = \bigsqcup_{i\in\omega} f(f^i(\bot)) = \bigsqcup_{i\in\omega} f^{i+1}(\bot) = \bigsqcup_{i\in\omega} f^i(\bot)$$

It is the least one. Suppose $f(x) = x$. Then $f^k(\bot) \sqsubseteq x$ by induction. So x is an upper bound for $\left(f^k(\bot)\right)_{i\in\omega}$. Hence $\bigsqcup_{i\in\omega} f^i(\bot) \sqsubseteq x$. □

Examples of ω-cpos

powersets $(\mathcal{P}(X), \subseteq)$. Least upper bounds = unions.

partial functions $(\mathbb{N} \rightharpoonup \mathbb{N}, \subseteq)$. A sub-cpo of $\mathcal{P}(\mathbb{N} \times \mathbb{N})$.

flat nats $\mathbb{N}_\perp \stackrel{\text{def}}{=} \{\perp\} \cup \mathbb{N}$. $x \sqsubseteq y \stackrel{\text{def}}{\equiv} x = \perp \vee x = y$

ω-chain are of two forms:

$$\perp \sqsubseteq \perp \sqsubseteq \perp \sqsubseteq \dots \text{ (with lub } \perp)$$

$$\perp \sqsubseteq \perp \sqsubseteq \cdots \sqsubseteq n \sqsubseteq n \sqsubseteq \dots \text{ (with lub } n)$$

streams $\Sigma^\infty \stackrel{\text{def}}{=}$ **finite or infinite sequences** over Σ. $w \sqsubseteq v$ iff w is a prefix of v. An ω-chain over $\Sigma = \mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}$:

$$\epsilon \sqsubseteq \langle 0 \rangle \sqsubseteq \langle 0, 0 \rangle \sqsubseteq \langle 0, 0, 0 \rangle \sqsubseteq \dots$$

Lub: the infinite sequence 0^ω.

Examples of monotonic and continuous functions

Flat booleans: $\mathbb{B}_\perp \stackrel{\text{def}}{=} \{\perp\} \cup \mathbb{B}$. $x \sqsubseteq y \stackrel{\text{def}}{\equiv} x = \perp \vee x = y$

Define three functions $f_1, f_2, f_3 : \mathbb{B}^\infty \to \mathbb{B}_\perp$.

$$f_1(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w \text{ contains a } 1 \\ \perp & \text{otherwise} \end{cases} \qquad f_2(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w \text{ contains a } 1 \\ 0 & \text{if } w = 0^\omega \\ \perp & \text{otherwise} \end{cases}$$

$$f_3(w) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } w \text{ contains a } 1 \\ 0 & \text{otherwise} \end{cases}$$

- f_1 is continuous: it examines the stream element-by-element.
- f_2 is monotonic but not continuous: it makes a decision by looking at the entirety of an infinite stream.
- f_3 is just awful.

Summary

Goal: mathematically defining functions by recursion.

Main ideas:

- recursive definitions as **fixed points**
- the **partial order of definedness**
- **least upper bounds** (lub) as limits/completed objects
- **monotonic** and **continuous** functions as (i) computational functions and (ii) acceptable templates for recursive definitions
- the **fixed point theorem**: constructing fixed points by iterating continuous functions (can be generalised: if just monotone, we can iterate transfinitely).

Beyond

- The semantics of **PCF**: simply-typed λ-calculus $+$ recursion.
- **Program logics** for recursion: computational induction.
- So far: **convergence**. But equally important is **approximation**. For example, partial functions are **algebraic**: $f = \bigcup_{\theta \subseteq_{\mathsf{fin}} f} \theta$. ω-cpos that are continuous, algebraic, ...
- Semantics of **recursive types**. In Haskell:

  ```
  data Tree = Leaf Int | Node Tree Int Tree
  ```

  Must construct a mathematical 'space' X that provides a solution to the **recursive domain equation**

  $$X \cong \mathbb{N}_\perp \oplus (X \times \mathbb{N}_\perp \times X)_\perp$$

- **Information Systems**: an equivalent presentation.
- **Synthetic Domain Theory**: a closer connection with computabiltity.

Synthetic Guarded Domain Theory

There is another way: **take step-indexing seriously**. Replace ω-cpos with *sets constructed over time*:

$$P = P(0) \xleftarrow{r_0} P(1) \xleftarrow{r_1} P(2) \xleftarrow{r_2} \dots$$

$P(i) =$ values at time i. $r_i : P(i+1) \to P(i)$ **trims** values.

Delaying a computation:

$$\blacktriangleright P = \{*\} \xleftarrow{!} P(0) \xleftarrow{r_0} P(1) \xleftarrow{r_1} P(2) \xleftarrow{r_2} \dots$$

A **causal** function $f : P \to Q$ consists of a family $f_i : P(i) \to Q(i)$ of functions that is 'compatible' with trimming.

Theorem
Every causal function $f : \blacktriangleright P \to P$ has a guarded fixed point.

Often just as good as domain theory. Excellent for recursive types!

References i

This presentation is based on lecture notes by Samson Abramsky. ($\approx$2007).

The history of the fixed point theorem:

- J.-L. Lassez, V.L. Nguyen, and E.a. Sonenberg (1982). "Fixed point theorems and semantics: a folk tale". In: *Information Processing Letters* 14.3, pp. 112–116. DOI: 10.1016/0020-0190(82)90065-5

Standard references on domain theory—a book and a survey:

- V. Stoltenberg-Hansen, I. Lindstrom, and E. R. Griffor (1994). *Mathematical Theory of Domains*. Cambridge: Cambridge University Press

- Samson Abramsky and Achim Jung (1994). "Domain Theory". In: *Handbook of Logic in Computer Science*. Ed. by Samson Abramsky, Dov M. Gabbay, and Thomas S. E. Maibaum. Vol. 3. Oxford University Press, pp. 1–168

Possibly the most clear and concise reference to PCF/LCF:

- Thomas Streicher (2006). *Domain-theoretic Foundations of Functional Programming*. World Scientific

A really unusual and fascinating book on (a) the connections of domain theory with topology, and (b) the intuitive meanings of many domain-theoretic and topological concepts in Haskell:

- M Escardo (Nov. 2004). "Synthetic Topology of Data Types and Classical Spaces". en. In: *Electronic Notes in Theoretical Computer Science* 87, pp. 21–156. DOI: 10.1016/S1571-0661(04)05135-7

A very similar blog post:

`http://math.andrej.com/2008/11/21/a-haskell-monad-for-infinite-search-in-finite-time/`

The source of all synthetic guarded domain theory:

- Lars Birkedal et al. (2012). "First steps in synthetic guarded domain theory: step-indexing in the topos of trees". In: *Logical Methods in Computer Science* 8.4. DOI: 10.2168/LMCS-8(4:1)2012