Lecture
Computability
(DIT312, DAT415)

Nils Anders Danielsson

2019-11-11

Today

Two models of computation:

- PRF.
- The recursive functions.

PRF

The primitive recursive functions

- A model of computation.
- Programs taking tuples of natural numbers to natural numbers.
- Every program is terminating.

Sketch

The primitive recursive functions can be constructed in the following ways:

$$f\left(\right) = 0$$

$$f\left(x\right) = 1 + x$$

$$f\left(x_1, ..., x_k, ..., x_n\right) = x_k$$

$$f\left(x_1, ..., x_n\right) = g\left(h_1\left(x_1, ..., x_n\right), ..., h_k\left(x_1, ..., x_n\right)\right)$$

$$\begin{aligned} f\left(x_1, ..., x_n, 0\right) &= g\left(x_1, ..., x_n\right) \\ f\left(x_1, ..., x_n, 1 + x\right) &= \\ &\quad h\left(x_1, ..., x_n, x, f\left(x_1, ..., x_n, x\right)\right) \end{aligned}$$

Abstract syntax

Vectors

Vectors, lists of a fixed length:

$$\frac{}{\textsf{nil} \in A^0} \qquad \frac{xs \in A^n \qquad x \in A}{xs, x \in A^{1+n}}$$

Read nil, x, y, z as $((\textsf{nil}, x), y), z$.

Indexing

An indexing operation can be defined by (a slight variant of) primitive recursion:

$$
\begin{aligned}
&\mathit{index} \in A^n \to \{\, i \in \mathbb{N} \mid 0 \leq i < n \,\} \to A \\
&\mathit{index}\,(xs, x)\ \mathsf{zero} \qquad\ = x \\
&\mathit{index}\,(xs, x)\ (\mathsf{suc}\ n) = \mathit{index}\ xs\ n
\end{aligned}
$$

Abstract syntax

PRF_n: Functions that take n arguments ($n \in \mathbb{N}$).

$$\frac{}{\mathsf{zero} \in PRF_0} \qquad \frac{}{\mathsf{suc} \in PRF_1}$$

$$\frac{i \in \mathbb{N} \qquad 0 \leq i < n}{\mathsf{proj}\ i \in PRF_n}$$

$$\frac{f \in PRF_m \qquad gs \in (PRF_n)^m}{\mathsf{comp}\ f\ gs \in PRF_n}$$

$$\frac{f \in PRF_n \qquad g \in PRF_{2+n}}{\mathsf{rec}\ f\ g \in PRF_{1+n}}$$

Denotational semantics

Denotational semantics

$$
\begin{aligned}
&[\![\_]\!] \in PRF_n \to (\mathbb{N}^n \to \mathbb{N})\\
&[\![\,\mathsf{zero}\,]\!]\ \mathsf{nil} = 0\\
&[\![\,\mathsf{suc}\,]\!]\ (\mathsf{nil}, n) = 1 + n\\
&[\![\,\mathsf{proj}\ i\,]\!]\ \rho = \mathit{index}\ \rho\ i\\
&[\![\,\mathsf{comp}\ f\ gs\,]\!]\ \rho = [\![f]\!]\ ([\![gs]\!]\star\ \rho)\\
&[\![\,\mathsf{rec}\ f\ g\,]\!]\ (\rho, \mathsf{zero}) = [\![f]\!]\ \rho\\
&[\![\,\mathsf{rec}\ f\ g\,]\!]\ (\rho, \mathsf{suc}\ n) = [\![g]\!]\ (\rho, n, [\![\mathsf{rec}\ f\ g]\!]\ (\rho, n))
\end{aligned}
$$

$$
\begin{aligned}
&[\![\_]\!]\star \in (PRF_m)^n \to (\mathbb{N}^m \to \mathbb{N}^n)\\
&[\![\,\mathsf{nil}\,]\!]\star\ \rho = \mathsf{nil}\\
&[\![\,fs, f\,]\!]\star\ \rho = [\![fs]\!]\star\ \rho, [\![f]\!]\ \rho
\end{aligned}
$$

Denotational semantics

$$
\begin{array}{lll}
[\![\_]\!] & \in PRF_n \to (\mathbb{N}^n \to \mathbb{N}) & \\
[\![\,\mathsf{zero}\,]\!]\ \mathsf{nil} & = 0 & \\
[\![\,\mathsf{suc}\,]\!]\ (\mathsf{nil}, n) & = 1 + n & \\
[\![\,\mathsf{proj}\ i\,]\!]\ \rho & = \mathit{index}\ \rho\ i & \\
[\![\,\mathsf{comp}\ f\ gs\,]\!]\ \rho & = [\![f]\!]\ ([\![gs]\!]\star\ \rho) & \\
[\![\,\mathsf{rec}\ f\ g\,]\!]\ (\rho, n) & = \mathit{rec}\ ([\![f]\!]\ \rho) & \\
 & \quad (\lambda\, n\ r.\ [\![g]\!]\ (\rho, n, r)) & \\
 & \quad n &
\end{array}
$$

$$
\begin{array}{l}
[\![\_]\!]\star \in (PRF_m)^n \to (\mathbb{N}^m \to \mathbb{N}^n) \\
[\![\,\mathsf{nil}\,]\!]\star\ \rho = \mathsf{nil} \\
[\![\,fs, f\,]\!]\star\ \rho = [\![fs]\!]\star\ \rho, [\![f]\!]\ \rho
\end{array}
$$

Quiz

Which of the following terms, all in PRF_2, define addition?

1. rec $($proj $0)$ $($proj $0)$
2. rec $($proj $0)$ $($proj $1)$
3. rec $($proj $0)$ $($comp suc $($nil, proj $0))$
4. rec $($proj $0)$ $($comp suc $($nil, proj $1))$

Hint: Examine $[\![p]\!]$ $($nil$, m, n)$ for each program p.

Addition

Goal: Define add satisfying the following equations:

$$\begin{aligned}
\forall\, m \in \mathbb{N}.\quad & [\![add]\!]\ (\mathsf{nil}, m, \mathsf{zero}) \ = m \\
\forall\, m, n \in \mathbb{N}.\ & [\![add]\!]\ (\mathsf{nil}, m, \mathsf{suc}\ n) = \\
& \qquad \mathsf{suc}\ ([\![add]\!]\ (\mathsf{nil}, m, n))
\end{aligned}$$

If we can find a definition of add that satisfies these equations, then we can use structural induction to prove that add is an implementation of addition.

Addition

Perhaps we can use rec:

$$\begin{aligned}
\forall\, m \in \mathbb{N}.\quad & [\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, \mathsf{zero}) \ = m \\
\forall\, m, n \in \mathbb{N}.\ & [\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, \mathsf{suc}\ n) = \\
& \qquad \mathsf{suc}\ ([\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, n))
\end{aligned}$$

Addition

Perhaps we can use rec:

$$
\begin{aligned}
\forall\, m \in \mathbb{N}.\quad & [\![f]\!]\ (\mathsf{nil}, m) && = m \\
\forall\, m, n \in \mathbb{N}.\ & [\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, \mathsf{suc}\ n) = \\
& \qquad \mathsf{suc}\ ([\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, n))
\end{aligned}
$$

Addition

Perhaps we can use rec:

$$\begin{aligned}
\forall\ m \in \mathbb{N}.\quad & [\![f]\!]\ (\mathsf{nil}, m) && = m \\
\forall\ m, n \in \mathbb{N}.\ & [\![g]\!]\ (\mathsf{nil}, m, n, [\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, n)) = \\
& \quad \mathsf{suc}\ ([\![\mathsf{rec}\ f\ g]\!]\ (\mathsf{nil}, m, n))
\end{aligned}$$

Addition

The zero case:

$$\forall\, m \in \mathbb{N}.\, [\![f]\!]\ (\mathsf{nil}, m) = m$$

Addition

The zero case:

$$\forall\ m \in \mathbb{N}.\ [\![\mathsf{proj}\ 0]\!]\ (\mathsf{nil}, m) = m$$

Addition

The suc case:

$$\forall\, m, n \in \mathbb{N}.\, [\![g]\!]\ (\mathsf{nil}, m, n, [\![\mathsf{rec}\, f\, g]\!]\ (\mathsf{nil}, m, n)) = \mathsf{suc}\ ([\![\mathsf{rec}\, f\, g]\!]\ (\mathsf{nil}, m, n))$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![g]\!]\ (\mathsf{nil}, m, n, r) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![\mathsf{comp}\ h\ hs]\!]\ (\mathsf{nil}, m, n, r) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![h]\!]\ ([\![hs]\!]\star\ (\mathsf{nil}, m, n, r)) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![\mathsf{suc}]\!]\ ([\![\mathsf{nil}, k]\!]\star\ (\mathsf{nil}, m, n, r)) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![\mathsf{suc}]\!]\ (\mathsf{nil}, [\![k]\!]\ (\mathsf{nil}, m, n, r)) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\, m, n, r \in \mathbb{N}.\, \mathsf{suc}\ (\llbracket k \rrbracket\ (\mathsf{nil}, m, n, r)) = \mathsf{suc}\ r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![k]\!]\ (\mathsf{nil}, m, n, r) = r$$

Addition

The suc case:

$$\forall\ m, n, r \in \mathbb{N}.\ [\![\mathsf{proj}\ 0]\!]\ (\mathsf{nil}, m, n, r) = r$$

Addition

We end up with the following definition:

$$\text{rec}\ (\text{proj}\ 0)\ (\text{comp}\ \text{suc}\ (\text{nil}, \text{proj}\ 0))$$

Big-step operational semantics

Big-step operational semantics

- The semantics can also be defined inductively.
- $f\,[\rho] \Downarrow n$ means that the result of evaluating f with input ρ is n.
- $f\,[\rho] \Downarrow n$ is well-formed ("type-correct") if

$$\exists\, m \in \mathbb{N}.\, f \in \mathit{PRF}_m \wedge \rho \in \mathbb{N}^m \wedge n \in \mathbb{N}.$$

- $fs\,[\rho] \Downarrow^\star \rho'$ is well-formed if

$$\begin{aligned}&\exists\, m, n \in \mathbb{N}.\\ &\quad f \in (\mathit{PRF}_m)^n \wedge \rho \in \mathbb{N}^m \wedge \rho' \in \mathbb{N}^n.\end{aligned}$$

- Note that well-formed statements do not need to be true.

Big-step operational semantics

$$\frac{}{\textsf{zero}\,[\textsf{nil}] \Downarrow 0} \qquad \frac{}{\textsf{suc}\,[\textsf{nil}, n] \Downarrow 1 + n}$$

$$\frac{}{\textsf{proj}\ i\,[\rho] \Downarrow \mathit{index}\,\rho\ i}$$

$$\frac{f\,[\rho] \Downarrow n}{\textsf{rec}\,f\,g\,[\rho, \textsf{zero}] \Downarrow n} \qquad \frac{\begin{array}{c}\textsf{rec}\,f\,g\,[\rho, m] \Downarrow n \\ g\,[\rho, m, n] \Downarrow o\end{array}}{\textsf{rec}\,f\,g\,[\rho, \textsf{suc}\ m] \Downarrow o}$$

Big-step operational semantics

$$\frac{gs\,[\rho] \Downarrow^\star \rho' \qquad f\,[\rho'] \Downarrow n}{\mathsf{comp}\ f\ gs\,[\rho] \Downarrow n}$$

$$\frac{}{\mathsf{nil}\,[\rho] \Downarrow^\star \mathsf{nil}} \qquad \frac{fs\,[\rho] \Downarrow^\star ns \qquad f\,[\rho] \Downarrow n}{fs, f\,[\rho] \Downarrow^\star ns, n}$$

Equivalence

$f\,[\rho] \Downarrow n$ iff $[\![f]\!]\,\rho = n$,
$fs\,[\rho] \Downarrow^{\star} \rho'$ iff $[\![fs]\!]_{\star}\,\rho = \rho'$.

This can be proved by induction on the structure of the semantics in one direction, and f/fs in the other.

Equivalence

Thus the operational semantics is total and deterministic:

- $\forall f\,\rho.\ \exists\, n.\, f\,[\rho] \Downarrow\, n$.
- $\forall f\,\rho\; m\; n$.
  $f\,[\rho] \Downarrow\, m$ and $f\,[\rho] \Downarrow\, n$ implies $m = n$.

Quiz

Which of the following propositions are true?

1. comp zero nil $[\text{nil}, 5, 7] \Downarrow 0$
2. comp suc $(\text{nil}, \text{proj } 0)$ $[\text{nil}, 5, 7] \Downarrow 6$
3. rec zero $(\text{proj } 0)$ $[\text{nil}, 2] \Downarrow 0$

(All three statements are well-formed.)

Computability for PRF

No self-interpreter

- Not every (Turing-) computable function is primitive recursive.
- Exercise: Define a computable function
  $code \in PRF_1 \to \mathbb{N}$
  with a computable left inverse.
- There is no program $eval \in PRF_1$ satisfying

$$\forall\, f \in PRF_1, n \in \mathbb{N}.$$
$$[\![eval]\!]\ (\mathsf{nil}, \ulcorner (f, n) \urcorner) = [\![f]\!]\ (\mathsf{nil}, n),$$

where $\ulcorner (f, n) \urcorner = 2^{code\, f}\, 3^n$.

No self-interpreter

Proof sketch:

- Define $g \in PRF_1$ by

  $$\text{comp suc}\ (\text{nil}, \text{comp}\ eval\ (\text{nil}, f)),$$

  where $[\![f]\!]\ (\text{nil}, n) = 2^n\, 3^n$.
- We get

  $$\begin{aligned}
  &[\![g]\!]\ (\text{nil}, code\ g) = \\
  &1 + [\![eval]\!]\ (\text{nil}, [\![f]\!]\ (\text{nil}, code\ g)) = \\
  &1 + [\![eval]\!]\ (\text{nil}, 2^{code\ g}\, 3^{code\ g}) = \\
  &1 + [\![eval]\!]\ (\text{nil}, \ulcorner (g, code\ g) \urcorner) = \\
  &1 + [\![g]\!]\ (\text{nil}, code\ g).
  \end{aligned}$$

Knuth's up-arrow

- Addition amounts to repeatedly taking the successor:

$$m + n = \overbrace{\text{suc } (\ldots(\text{suc } m)\ldots)}^{n}$$

- Multiplication is repeated addition:

$$mn = \overbrace{m + \cdots + m}^{n}$$

- Exponentiation is repeated multiplication:

$$m^n = \overbrace{m \cdots m}^{n}$$

Knuth's up-arrow

We can continue:

$$
\begin{aligned}
m \uparrow\uparrow n &= \overbrace{m^{\cdot^{\cdot^{\cdot^{m}}}}}^{n} \\
m \uparrow\uparrow\uparrow n &= \overbrace{m \uparrow\uparrow (\cdots(m \uparrow\uparrow m)\cdots)}^{n} \\
m \uparrow\uparrow\uparrow\uparrow n &= \overbrace{m \uparrow\uparrow\uparrow (\cdots(m \uparrow\uparrow\uparrow m)\cdots)}^{n} \\
&\vdots
\end{aligned}
$$

All of these functions are primitive recursive.

Quiz

What is the value of $2 \uparrow\uparrow\uparrow 3$?

$$m \uparrow\uparrow n = \overbrace{m^{\cdot^{\cdot^{\cdot^{m}}}}}^{n}$$

$$m \uparrow\uparrow\uparrow n = \overbrace{m \uparrow\uparrow (\cdots(m \uparrow\uparrow m)\cdots)}^{n}$$

Knuth's up-arrow

A generalisation:

$$
\begin{aligned}
&\uparrow \in \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&m \uparrow^{\mathsf{zero}} \;\; k \qquad\; = mk \\
&m \uparrow^{\mathsf{suc}\ n} \mathsf{zero} \;\; = 1 \\
&m \uparrow^{\mathsf{suc}\ n} \mathsf{suc}\ k = m \uparrow^{n} (m \uparrow^{\mathsf{suc}\ n} k)
\end{aligned}
$$

This is a variant of Knuth's up-arrow notation.

Knuth's up-arrow

- Every individual function $\uparrow^n$ is primitive recursive.
- However, $\uparrow$ is not, even though it is computable.

The Ackermann function

- Another example of a computable function that is not primitive recursive.
- One variant:

$$\begin{aligned}
&ack \in \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\
&ack\,(\mathsf{zero}, \quad n) &&= \mathsf{suc}\ n \\
&ack\,(\mathsf{suc}\ m, \mathsf{zero}) &&= ack\,(m, \mathsf{suc}\ \mathsf{zero}) \\
&ack\,(\mathsf{suc}\ m, \mathsf{suc}\ n) &&= ack\,(m, ack\,(\mathsf{suc}\ m, n))
\end{aligned}$$

- The function "grows faster" than every primitive recursive function.

The recursive functions

The recursive functions

- A model of computation.
- Programs taking tuples of natural numbers to natural numbers.
- Not every program is terminating.

Abstract syntax

- Extends PRF with one additional constructor.
- RF_n: Functions that take n arguments.
- Minimisation:

$$\frac{f \in RF_{1+n}}{\textsf{min}\ f \in RF_n}$$

- Rough idea: min $f\,[\rho]$ is the smallest n for which $f\,[\rho, n]$ is 0.
- Note that there may not be such a number.

Big-step operational semantics

The operational semantics is extended:

$$\frac{f[\rho, n] \Downarrow 0 \qquad \forall m < n.\ \exists\, k \in \mathbb{N}.\, f[\rho, m] \Downarrow 1 + k}{\mathsf{min}\ f[\rho] \Downarrow n}$$

Big-step operational semantics

The operational semantics is extended:

$$\frac{f[\rho, n] \Downarrow 0 \qquad \forall m < n.\ \exists\, k \in \mathbb{N}.\, f[\rho, m] \Downarrow 1 + k}{\text{min } f[\rho] \Downarrow n}$$

The semantics is deterministic, but not total:

- $f[\rho] \Downarrow m$ and $f[\rho] \Downarrow n$ implies $m = n$.
- $\forall m.\ \exists\, f \in RF_m.\ \forall\, \rho.\ \nexists\, n.\, f[\rho] \Downarrow n$.

Quiz

- Construct $f \in RF_0$ in such a way that $\nexists n. f\,[\mathsf{nil}] \Downarrow n$.

Denotational semantics?

We can try to extend the denotational semantics:

$$\llbracket \_ \rrbracket \in RF_n \to (\mathbb{N}^n \to \mathbb{N})$$
$$\vdots$$
$$\llbracket \mathsf{min}\ f \rrbracket\ \rho = search\, f\, \rho\ 0$$

$$search \in RF_{1+n} \to \mathbb{N}^n \to \mathbb{N} \to \mathbb{N}$$
$$\begin{aligned} & search\, f\, \rho\ n = \\ & \quad \textbf{if} \quad \llbracket f \rrbracket\ (\rho, n) = 0 \\ & \quad \textbf{then}\ n \\ & \quad \textbf{else} \quad search\, f\, \rho\ (1 + n) \end{aligned}$$

Partial functions

- This "definition" does not give rise to (total) functions.
- We can instead define a semantics as a function to partial functions:

$$
\begin{aligned}
&[\![\_]\!] \in RF_n \to (\mathbb{N}^n \rightharpoonup \mathbb{N}) \\
&[\![f]\!]\ \rho = \\
&\quad \textbf{if} \quad\ \ f\,[\rho] \Downarrow n \text{ for some } n \\
&\quad \textbf{then}\ n \\
&\quad \textbf{else}\ \ \text{undefined}
\end{aligned}
$$

Expressiveness

- Equivalent to Turing machines, λ-calculus, ...

Summary

Two models of computation:

- PRF.
- The recursive functions.