Lecture
Models of computation
(DIT311, TDA184)

Nils Anders Danielsson

2018-11-12

Today

- Inductive definitions:
  - Functions defined by primitive recursion.
  - Proofs by structural induction.
- Two models of computation:
  - PRF.
  - The recursive functions. (If we have time.)

Natural numbers

The natural numbers

The set of natural numbers, $\mathbb{N}$, is defined inductively in the following way:

- zero $\in \mathbb{N}$.
- If $n \in \mathbb{N}$, then suc $n \in \mathbb{N}$.

The natural numbers

We can construct natural numbers by using these rules a finite number of times. Examples:

- $0 =$ zero.
- $1 =$ suc zero.
- $2 =$ suc $($suc zero$)$.

The value zero and the function suc are called *constructors*.

The natural numbers

An alternative way to present the rules:

$$\frac{}{\mathsf{zero} \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\mathsf{suc}\ n \in \mathbb{N}}$$

Propositions, predicates and relations

- A *proposition* is something that can (perhaps) be proved or disproved.
- A *predicate* on a set A is a function from A to propositions.
- A *binary relation* on two sets A and B is a function from A and B to propositions.
- Relations can also have more arguments.

Equality

Two natural numbers are equal if they are built up by the same constructors.

We can see this as an inductively defined relation:

$$\frac{}{\mathsf{zero} = \mathsf{zero}} \qquad \frac{m = n}{\mathsf{suc}\ m = \mathsf{suc}\ n}$$

(The names of the constructors have been omitted.)

Primitive recursion

We can define a function from $\mathbb{N}$ to a set A in the following way:

- A value $z \in A$, the function's value for zero.
- A function $s \in \mathbb{N} \to A \to A$, that given $n \in \mathbb{N}$ and the function's value for n gives the function's value for suc n.

Primitive recursion

A definition by primitive recursion can be given the following schematic form:

$$
\begin{aligned}
&f \in \mathbb{N} \to A \\
&f\,\mathsf{zero} \quad\;\; = z \\
&f\,(\mathsf{suc}\ n) = s\ n\ (f\ n)
\end{aligned}
$$

Primitive recursion

We can capture this scheme with a higher-order function:

$$
\begin{aligned}
&rec \in A \to (\mathbb{N} \to A \to A) \to \mathbb{N} \to A \\
&rec\ z\ s\ \mathsf{zero} = z \\
&rec\ z\ s\ (\mathsf{suc}\ n) = s\ n\ (rec\ z\ s\ n)
\end{aligned}
$$

Example: Equality with zero

- Can we define $\mathit{is\text{-}zero} \in \mathbb{N} \to \mathit{Bool}$ using primitive recursion?
- Let "A" be Bool.
- Scheme:

$$
\begin{aligned}
&\mathit{is\text{-}zero} \in \mathbb{N} \to \mathit{Bool} \\
&\mathit{is\text{-}zero}\ \mathsf{zero} \qquad = ? \\
&\mathit{is\text{-}zero}\ (\mathsf{suc}\ n) = ?
\end{aligned}
$$

Example: Equality with zero

- Can we define $is\text{-}zero \in \mathbb{N} \to Bool$ using primitive recursion?
- Let "A" be $Bool$.
- Scheme:

$$
\begin{aligned}
&is\text{-}zero \in \mathbb{N} \to Bool \\
&is\text{-}zero\ \mathsf{zero} &&= \mathsf{true} \\
&is\text{-}zero\ (\mathsf{suc}\ n) &&= \mathsf{false}
\end{aligned}
$$

Example: Equality with zero

- Can we define $\mathit{is\text{-}zero} \in \mathbb{N} \rightarrow \mathit{Bool}$ using primitive recursion?
- Let "A" be Bool.
- With the higher-order function:

$$\begin{aligned} &\mathit{is\text{-}zero} \in \mathbb{N} \rightarrow \mathit{Bool} \\ &\mathit{is\text{-}zero} = \mathit{rec}\ \mathsf{true}\ (\lambda\, n\ r.\, \mathsf{false}) \end{aligned}$$

Example: Addition

- Can we define $add \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ using primitive recursion?
- Let "A" be $\mathbb{N} \to \mathbb{N}$.
- Scheme:

$$
\begin{aligned}
& add \in \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
& add\, \mathsf{zero} \qquad\quad = ? \\
& add\, (\mathsf{suc}\ m) = ?
\end{aligned}
$$

Example: Addition

- Can we define $add \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ using primitive recursion?
- Let "A" be $\mathbb{N} \to \mathbb{N}$.
- Scheme:

$$
\begin{aligned}
&add \in \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
&add\,\mathsf{zero} \qquad\; = \lambda\, n.\, n \\
&add\,(\mathsf{suc}\ m) = ?
\end{aligned}
$$

Example: Addition

- Can we define $add \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ using primitive recursion?
- Let "A" be $\mathbb{N} \to \mathbb{N}$.
- Scheme:

$$\begin{aligned}
&add \in \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
&add\,\mathsf{zero} \qquad\quad = \lambda\, n.\, n \\
&add\,(\mathsf{suc}\ m) = \lambda\, n.\ ?
\end{aligned}$$

Example: Addition

- Can we define $add \in \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ using primitive recursion?
- Let "A" be $\mathbb{N} \to \mathbb{N}$.
- Scheme:

$$
\begin{aligned}
&add \in \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \\
&add\,\mathsf{zero} \qquad\quad = \lambda\, n.\, n \\
&add\,(\mathsf{suc}\ m) = \lambda\, n.\, \mathsf{suc}\ (add\ m\ n)
\end{aligned}
$$

Quiz

Which of the following terms define addition?

- $rec\ (\lambda\, n.\, n)\ (\lambda\, m\ r.\, \lambda\, n.\, \mathsf{suc}\ (r\ m\ n))$
- $rec\ (\lambda\, n.\, n)\ (\lambda\, m\ r.\, \lambda\, n.\, \mathsf{suc}\ (r\ n))$
- $rec\ (\lambda\, n.\, n)\ (\lambda\, m\ r.\, \lambda\, n.\, \mathsf{suc}\ (r\ m))$

Structural induction

Let us assume that we have a predicate P on $\mathbb{N}$. If we can prove the following two statements, then we have proved $\forall n.\, P\ n$:

- P zero.
- $\forall n.\, P\ n$ implies $P\ (\text{suc}\ n)$.

Example: Addition

Theorem: $\forall m \in \mathbb{N}.\ add\ m\ \mathsf{zero} = m$.

Proof:

- Let us use structural induction, with the predicate $P = \lambda\, m.\ add\ m\ \mathsf{zero} = m$.
- There are two cases:

$$
\begin{aligned}
& P\ \mathsf{zero} && \Leftarrow \{\text{By definition.}\} \\
& add\ \mathsf{zero}\ \mathsf{zero} = \mathsf{zero} && \Leftarrow \{\text{By definition.}\} \\
& \mathsf{zero} = \mathsf{zero}
\end{aligned}
$$

Example: Addition

Theorem: $\forall m \in \mathbb{N}.\ add\ m\ \mathsf{zero} = m$.

Proof:

- Let us use structural induction, with the predicate $P = \lambda\, m.\ add\ m\ \mathsf{zero} = m$.
- There are two cases:

$$
\begin{array}{ll}
P\ (\mathsf{suc}\ m) & \Leftarrow \\
add\ (\mathsf{suc}\ m)\ \mathsf{zero} = \mathsf{suc}\ m & \Leftarrow \\
\mathsf{suc}\ (add\ m\ \mathsf{zero}) = \mathsf{suc}\ m & \Leftarrow \\
add\ m\ \mathsf{zero} = m & \Leftarrow \\
P\ m &
\end{array}
$$

More inductively defined sets

Cartesian products

The cartesian product of two sets A and B is defined inductively in the following way:

$$\frac{x \in A \qquad y \in B}{\textsf{pair}\ x\ y \in A \times B}$$

Notice that this definition is "non-recursive".

Primitive recursion

Scheme for primitive recursion for pairs:

$$f \in A \times B \rightarrow C$$
$$f\,(\mathsf{pair}\ x\ y) = p\ x\ y$$

The corresponding higher-order function:

$$uncurry \in (A \rightarrow B \rightarrow C) \rightarrow A \times B \rightarrow C$$
$$uncurry\ p\ (\mathsf{pair}\ x\ y) = p\ x\ y$$

Structural induction

Let us assume that we have a predicate P on $A \times B$. If we can prove the following statement, then we have proved $\forall p.\, P\ p$:

- $\forall x\ y.\, P\,(\text{pair}\ x\ y)$.

Lists

The set of finite lists containing natural numbers is defined inductively in the following way:

$$\frac{}{\textsf{nil} \in \mathit{Nat\text{-}list}} \qquad \frac{x \in \mathbb{N} \qquad xs \in \mathit{Nat\text{-}list}}{\textsf{cons}\ x\ xs \in \mathit{Nat\text{-}list}}$$

Primitive recursion

Scheme for primitive recursion for natural number lists:

$$
\begin{aligned}
& f \in \mathit{Nat\text{-}list} \to A \\
& f\ \mathsf{nil} = n \\
& f\ (\mathsf{cons}\ x\ xs) = c\ x\ xs\ (f\ xs)
\end{aligned}
$$

The corresponding higher-order function:

$$
\begin{aligned}
& \mathit{listrec} \in A \to (\mathbb{N} \to \mathit{Nat\text{-}list} \to A \to A) \to \\
& \qquad \mathit{Nat\text{-}list} \to A \\
& \mathit{listrec}\ n\ c\ \mathsf{nil} = n \\
& \mathit{listrec}\ n\ c\ (\mathsf{cons}\ x\ xs) = c\ x\ xs\ (\mathit{listrec}\ n\ c\ xs)
\end{aligned}
$$

Structural induction

Let us assume that we have a predicate P on $Nat\text{-}list$. If we can prove the following statements, then we have proved $\forall xs.\, P\ xs$:

- P nil.
- $\forall x\ xs.\, P\ xs$ implies $P\ (\text{cons}\ x\ xs)$.

Pattern

- Given an inductive definition of the kind presented here, we can derive:
  - The structural induction principle.
  - The primitive recursion scheme.
- Pattern:
  - One case per constructor.
  - One argument per constructor argument, plus an extra argument (for induction: an inductive hypothesis) per *recursive* constructor argument.

Quiz

Define the booleans inductively. How many cases does the structural induction principle have?

- 1
- 2
- 3
- 4

Bonus question: Can you think of an inductive definition for which the answer would be 0?

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:

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

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

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

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

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

Vectors

Vectors, lists of a fixed length:

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

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

Indexing

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

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

Abstract syntax

PRF_n: Functions that take n arguments.

$$\frac{}{\mathsf{zero} \in PRF_0} \qquad \frac{}{\mathsf{suc} \in PRF_1} \qquad \frac{0 \leq i < n}{\mathsf{proj}\ i \in PRF_n}$$

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

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

Denotational semantics

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

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

Denotational semantics

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

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

Quiz

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

- rec $($proj $0)$ $($proj $0)$
- rec $($proj $0)$ $($proj $1)$
- rec $($proj $0)$ $($comp suc $($nil, proj $0))$
- 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:

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

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

Addition

Perhaps we can use rec:

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

Addition

Perhaps we can use rec:

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

Addition

Perhaps we can use rec:

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

Addition

The zero case:

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

Addition

The zero case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

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

Addition

The suc case:

$$\forall\, m\; n\; r.\; [\![\mathsf{proj}\ 1]\!]\ (\mathsf{nil}, m, r, n) = r$$

Addition

We end up with the following definition:

$$\text{rec } (\text{proj } 0)\ (\text{comp suc } (\text{nil}, \text{proj } 1))$$

Big-step operational semantics

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

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

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

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

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

Big-step operational semantics

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

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

$$\frac{f\,[\rho] \Downarrow n}{\mathsf{rec}\,f\,g\,[\rho, \mathsf{zero}] \Downarrow n} \qquad \frac{\mathsf{rec}\,f\,g\,[\rho, m] \Downarrow n \qquad g\,[\rho, n, m] \Downarrow o}{\mathsf{rec}\,f\,g\,[\rho, \mathsf{suc}\ m] \Downarrow o}$$

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?

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

No self-interpreter

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

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

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

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

The Ackermann function

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

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

- For more details, see Nordström, *The primitive recursive functions*.

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:

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

- 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:

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

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:

$$
\begin{aligned}
&[\![\_]\!] \in RF_n \to (\mathbb{N}^n \to \mathbb{N}) \\
&\vdots \\
&[\![\mathsf{min}\ f]\!]\ \rho = \mathit{search}\, f\, \rho\ 0
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{search} \in RF_{1+n} \to \mathbb{N}^n \to \mathbb{N} \to \mathbb{N} \\
&\mathit{search}\, f\, \rho\ n = \\
&\quad \mathbf{if}\quad [\![f]\!]\ (\rho, n) = 0 \\
&\quad \mathbf{then}\ n \\
&\quad \mathbf{else}\ \ \mathit{search}\, f\, \rho\ (1 + n)
\end{aligned}
$$

Partial functions

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

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

Expressiveness

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

Summary

- Inductive definitions:
  - Functions defined by primitive recursion.
  - Proofs by structural induction.
- Two models of computation:
  - PRF.
  - The recursive functions.