Lecture
Computability
(DIT312, DAT415)

Nils Anders Danielsson

2019-11-06

Today

- Inductively defined sets.
- Functions defined by primitive recursion.
- Proofs by structural induction.

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:

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

Primitive recursion

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

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

Example: Equality with zero

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

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

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

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

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:

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

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:

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

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:

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

Quiz

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

Which of the following terms define addition?

1. $rec\ (\lambda\, n.\ n)\ (\lambda\, m\ r.\ \lambda\, n.\ \mathsf{suc}\ (r\ m\ n))$
2. $rec\ (\lambda\, n.\ n)\ (\lambda\, m\ r.\ \lambda\, n.\ \mathsf{suc}\ (r\ n))$
3. $rec\ (\lambda\, n.\ n)\ (\lambda\, m\ r.\ \lambda\, n.\ \mathsf{suc}\ (r\ m))$

Addition again

Another way to define addition:

- Let us fix $m \in \mathbb{N}$.
- Now we can define "addition by m".
- Let "A" be $\mathbb{N}$.
- Scheme:

$$\begin{aligned} add'_m &\in \mathbb{N} \to \mathbb{N} \\ add'_m\ \mathsf{zero} &= m \\ add'_m\ (\mathsf{suc}\ n) &= \mathsf{suc}\ (add'_m\ n) \end{aligned}$$

Addition again

Another way to define addition:

- Scheme:

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

- Using rec:

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

Quiz

Multiplication by m, defined recursively:

$$
\begin{aligned}
&mul \in \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&mul\; m\; \mathsf{zero} = \mathsf{zero} \\
&mul\; m\; (\mathsf{suc}\; n) = add\; m\; (mul\; m\; n)
\end{aligned}
$$

Complete the following definition of multiplication. You can make use of addition (add).

$mul\; m = rec\,?\,(\lambda\, n\; r.\, ?)$

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 $\forall n \in \mathbb{N}.\ P\ n$:

- P zero.
- $\forall n \in \mathbb{N}.\ P\ n$ 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:

$$
\begin{aligned}
&f \in A \times B \to C \\
&f\,(\mathsf{pair}\ x\ y) = p\ x\ y
\end{aligned}
$$

The corresponding higher-order function:

$$
\begin{aligned}
&uncurry \in (A \to B \to C) \to A \times B \to C \\
&uncurry\ p\ (\mathsf{pair}\ x\ y) = p\ x\ y
\end{aligned}
$$

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 $\forall p \in A \times B.\, P\, p$:

- $\forall x \in A.\ \forall\, y \in B.\, P\, (\mathsf{pair}\ x\ y)$.

Monomorphic lists

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

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

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:

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

Note that the recursion does not descend into the natural numbers.

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 $\forall xs \in Nat\text{-}list.\ P\ xs$:

- P nil.
- $\forall x \in \mathbb{N}.\ \forall\ xs \in Nat\text{-}list.$
  $P\ xs$ implies $P\ (\text{cons}\ x\ xs)$.

Lists

The set of finite lists containing elements of the set A is defined inductively in the following way:

$$\frac{}{\mathsf{nil} \in \mathit{List}\ A} \qquad \frac{x \in A \qquad xs \in \mathit{List}\ A}{\mathsf{cons}\ x\ xs \in \mathit{List}\ A}$$

Primitive recursion

Scheme for primitive recursion for lists:

$$\begin{aligned}
&f \in \mathit{List}\ A \to B \\
&f\,\mathsf{nil} && = n \\
&f\,(\mathsf{cons}\ x\ xs) && = c\ x\ xs\ (f\ xs)
\end{aligned}$$

The corresponding higher-order function:

$$\begin{aligned}
&\mathit{listrec} \in B \to (A \to \mathit{List}\ A \to B \to B) \to \\
&\qquad\qquad \mathit{List}\ A \to B \\
&\mathit{listrec}\ n\ c\ \mathsf{nil} && = n \\
&\mathit{listrec}\ n\ c\ (\mathsf{cons}\ x\ xs) && = c\ x\ xs\ (\mathit{listrec}\ n\ c\ xs)
\end{aligned}$$

Structural induction

Let us assume that we have a predicate P on $List\ A$. If we can prove the following statements, then we have proved $\forall xs \in List\ A.\ P\ xs$:

- P nil.
- $\forall x \in A.\ \forall\ xs \in List\ A.$
  $P\ xs$ implies P (cons $x\ xs$).

Quiz

Use $listrec$ and $uncurry$ to define a function from $List\,(A \times B)$ to $List\,B$ that replaces every pair in the list with its second component.

$listrec\,?\,?$

Quiz

Use $listrec$ and $uncurry$ to define a function from $List\,(A \times B)$ to $List\,B$ that replaces every pair in the list with its second component.

$listrec$ nil $(\lambda\, p\ ps\ r.\,$ cons $(uncurry\ (\lambda\, x\ y.\, ?)\ p) ?)$

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.

Pattern

Pattern (with recursive constructor arguments last):

$$
\begin{array}{l}
drec \in \text{One assumption per constructor} \to D \to A \\
drec\, f_1 \dots f_k\ (\mathsf{c}_1\ x_1 \dots x_{n_1}) = \\
\quad f_1\ x_1 \dots x_{n_1}\ (drec\, f_1 \dots f_k\ x_{i_1}) \dots (drec\, f_1 \dots f_k\ x_{n_1}) \\
\vdots \\
drec\, f_1 \dots f_k\ (\mathsf{c}_k\ x_1 \dots x_{n_k}) = \\
\quad f_k\ x_1 \dots x_{n_k}\ (drec\, f_1 \dots f_k\ x_{i_k}) \dots (drec\, f_1 \dots f_k\ x_{n_k})
\end{array}
$$

Quiz

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

- 1
- 2
- 3
- 4

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

Summary

- Inductively defined sets.
- Functions defined by primitive recursion.
- Proofs by structural induction.