Lecture 20: Primitive Recursion

Conditional Functions

MATH230

Te Kura Pāngarau | School of Mathematics and Statistics
Te Whare Wānanga o Waitaha | University of Canterbury

Outline

Piecewise Functions

If the primitive recursive functions are to be able to perform all possible computations, then we need to be able to write functions that can deal with conditional branching.

$$f(n) = \begin{cases} 3n+1 & n \text{ odd} \\ \frac{n}{2} & n \text{ even} \end{cases}$$

Addition of Recursive Functions

Given two functions f_1, f_2, we can define the point-wise sum of these functions as follows

$$(f_1 + f_2)(n) = f_1(n) + f_2(n)$$

Addition of Recursive Functions

Given functions f_1, f_2, f_3, we can define the point-wise sum of these functions as follows

$$(f_1 + f_2 + f_3)(n) = f_1(n) + f_2(n) + f_3(n)$$

Addition of Recursive Functions

Given a primitive recursive function f, we can define the point-wise sum of the function with itself finitely many times as follows

$$(f + \cdots + f)(n) = f(n) + \cdots + f(n)$$

Multiplication of Recursive Functions

Given two functions f_1, f_2, we can define the point-wise multiplication of these functions as follows

$$(f_1 \cdot f_2)(n) = f_1(n) \cdot f_2(n)$$

Multiplication of Recursive Functions

Given functions f_1, f_2, f_3, we can define the point-wise product of these functions as follows

$$(f_1 \times f_2 \times f_3)(n) = f_1(n) \times f_2(n) \times f_3(n)$$

Multiplication of Recursive Functions

Given a primitive recursive function f, we can define the point-wise product of the function with itself finitely many times as follows

$$(f \times \cdots \times f)(n) = f(n) \times \cdots \times f(n)$$

Recursive Predicates

In order to write conditional functions, we need to be able to compute whether the conditions are satisfied i.e. we need primitive recursive definitions of the conditions.

We define the characteristic function of an *n*-ary predicate $P(\mathsf{x})$

$$\chi_P(\mathsf{x}) := \begin{cases} 1 & \text{if } P(\mathsf{x}) \\ 0 & \text{if } \neg P(\mathsf{x}) \end{cases}$$

We say an *n*-ary predicate is primitive recursive if χ_P is primitive recursive.

Primitive Recursive Predicates

Conditional programs (piece-wise functions) may branch according to the values of the following predicates

- Less than, less than or equal to (order)
- Greater than, greater than or equal to (order)
- Equal (identity)

Propositional Connectives

Suppose $P(\mathsf{x})$ and $Q(\mathsf{x})$ are two primitive n-ary recursive predicates.

Show that the following are all primitive recursive

$\neg P(\mathsf{x})$
$P(\mathsf{x}) \vee Q(\mathsf{x})$
$P(\mathsf{x}) \wedge Q(\mathsf{x})$
$P(\mathsf{x}) \rightarrow Q(\mathsf{x})$

Details left for tutorial.

Conditional Functions

We may combine the addition, multiplication, and primitive recursive predicates to obtain conditional functionality.

Conditional Functions

Suppose we have k distinct n-ary primitive recursive predicates $A_1, A_2, \ldots, A_k$ such that each x satisfies precisely one of them. We can use such a family of predicates to define piece-wise functions

$$g(\mathrm{x}) := \chi_{A_1}(\mathrm{x}) \cdot f_1(\mathrm{x}) + \chi_{A_2}(\mathrm{x}) \cdot f_2(\mathrm{x}) + \cdots + \chi_{A_k}(\mathrm{x}) \cdot f_k(\mathrm{x})$$

Existential Quantifier

If $Q(x)$ is a primitive recursive predicate, then bounded search for an integer that satisfies it is primitive recursive. That is to say, the following is primitive recursive.

$$\chi_{\exists y<nQ(y)} = \begin{cases} 1 & \text{if there exists } y < n \text{ such that } Q(y). \\ 0 & \text{otherwise.} \end{cases}$$

Bounded existential quantification is primitive recursive.

Example

$$\mathrm{div}(x, y) = \begin{cases} 1 & \text{if } x \mid y. \\ 0 & \text{otherwise.} \end{cases}$$

Details left for tutorial.

Universal Quantifier

If $Q(x)$ is a primitive recursive predicate, then checking if all integers less than a specified bound satisfy it is a primitive recursive process.

$$\chi_{\forall y<nQ(y)} = \begin{cases} 1 & \text{if all } y < n \text{ satisfy } Q(y). \\ 0 & \text{otherwise.} \end{cases}$$

Bounded universal quantification is primitive recursive.

Example

$$\text{prime?}(x) = \begin{cases} 1 & \text{if } x \text{ is prime.} \\ 0 & \text{otherwise.} \end{cases}$$

Details left for tutorial.

Primitive Recursion Language

In a sense we are defining a (programming?) language for a certain class of functions. We are allowed to use the following functions and constructions to build new ones:

Constants

Projections

Arithmetic functions

Sums and products of functions

$\sum_{y<n}$ and $\prod_{y<n}$

Predicates

Propositional logic

Bounded quantification

Which mathematical questions can be answered with such functions?

Further Reading

Here are some recommended reading to follow up on the lecture content.

- SEP: Recursive Functions.
- Computability, Richard Epstein.
- Computability Theory, Enderton.
- Lectures on the Philosophy of Mathematics, Joel Hamkins.