Lecture Contents

- What is a primitive recursive function?
- How to prove results about all p.r. functions
- The p.r. functions are computable . . .
- . . . but not all computable functions are p.r.
- Recursive functions
- The idea of a characteristic function, which enables us to define . . .
- . . . the idea of recursive properties and relations.
Arithmetic involves more than just addition and multiplication, e.g. exponentiation.

We are going to look at an important subclass of the computable functions, the primitive recursive functions.

Then we will look at the recursive functions.

It can be shown that $L_A$, the language of basic arithmetic, can express all recursive functions and relations.
Story to come

- Q and hence PA can capture all recursive functions and relations.
- We will then talk in more detail about Goedel coding.
- The relation Prf for Q and PA is recursive (p.r. in fact) and we use that to construct a Goedel sentence.
Some definitions by primitive recursion

\[
\begin{align*}
  +(x, 0) &= x \\
  +(x, s(n)) &= s(+(x, n))
\end{align*}
\]

\[
\begin{align*}
  \times(x, 0) &= 0 \\
  \times(x, s(n)) &= +(\times(x, n), x)
\end{align*}
\]

\[
\begin{align*}
  f(0) &= 1 \\
  f(s(n)) &= f(n) \times s(n)
\end{align*}
\]
Some definitions by primitive recursion

\[
\begin{align*}
f(x, 0) &= 1 \\
f(x, s(n)) &= f(x, n) \times x \\
p(0) &= 0 \quad z(0) = 1 \\
p(s(n)) &= n \quad z(s(n)) = 0 \\
e(x, 0) &= z(x) \\
e(x, s(n)) &= z(z(x))) \times e(p(x), n)
\end{align*}
\]
Roughly: a primitive recursive function is one that can be similarly characterised using a chain of definitions by recursion and composition.

A little more specifically

\[
\begin{align*}
  f(0) &= g \\
  f(s(z)) &= h(z, f(z)) \\
  f(x, 0) &= g(x) \\
  f(x, s(z)) &= h(x, z, f(x, z))
\end{align*}
\]
Fully generally, if $g$ and $h$ are p.r. then $f$ defined by:

$$f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$$
$$f(\overrightarrow{x}, s(z)) = h(\overrightarrow{x}, z, f(\overrightarrow{x}, z))$$

is said to be *defined by primitive recursion* in terms of $g$ and $h$. 
But the definitions above don’t all fit this pattern, to get started we also need:

1. the successor function: \( s \).
2. the zero constant function: \( 0 \).
3. for any \( k \), the \( i \)-th projection function on \( k \):

\[
l_k^i(x_1 \ldots x_k) = x_i
\]

4. to be able to compose p.r. functions.
So the p.r. definition of addition becomes:

\[
\begin{align*}
+&(x, 0) = l_1^1(x) \\
+(x, s(n)) &= sl_3^3(x, n, +(x, n))
\end{align*}
\]
Full definitions

- If $g$ is $n$-ary and $h_1 \ldots h_n$ are $m$-ary, then an $m$-ary function $f(x_1 \ldots x_m) = g(h_1(x_1 \ldots x_m), \ldots, h_n(x_1 \ldots x_m))$ is said to be defined by composition in terms of $g$ and $h$.

- 1. $s$, 0 and $l_i^k$ are primitive recursive (for any $i, k$ s.t. $1 \leq i \leq k$)
   2. If $g$ and $h_1 \ldots h_n$ are p.r. and $f$ is defined by composition in terms of $g$ and the $h_i$ then $f$ is p.r.
   3. If $g$ and $h$ are p.r. and $f$ is defined by primitive recursion in terms of $g$ and $h$, then $f$ is p.r.

- It is useful to identify a p.r. function with its history.
A full definition for the p.r. function $f$ is a specification of a sequence of functions $f_0, f_1, f_2, \ldots, f_k$ where each $f_j$ is either an initial function or is defined from previous functions in the sequence by composition or recursion, and $f_k = f$.

We can now prove facts about all primitive functions by induction on their full definition (like proofs by induction on derivations).
The p.r. functions are computable

1. The initial functions are computable.
2. Compositions of computable functions are computable.
3. Functions defined by primitive recursion from computable functions are computable.

For example, to compute $f(\overrightarrow{x}, n)$ where $f$ is defined by primitive recursion from computable $g$ and $h$, we can apply the primitive recursive clause $n + 1$ times and then compute with only $h$ and $g$. 
Not all computable functions are p.r.

- There are effectively computable numerical functions which are not primitive recursive.
- The p.r. functions are effectively enumerable: \( f_1, f_2 \ldots \)
  \[
  \delta(n) = f_n(n) + 1
  \]
- If \( \delta \) is p.r. then it is \( f_d \), but then:
  \[
  f_d(d) = \delta(d) = f_d(d) + 1
  \]
Not all computable functions are p.r.

- There can be no effective listing of all the intuitively computable total functions.
- Compare with the diagonalisation argument. Does this imply that there are uncountably many computable functions?
- $\delta$ is not p.r. as we cannot put an upper bound on how much computation is involved in determining $f_n(n)$ for each $n$.
- Note that for each $n$ there is a p.r. function that calculates $\delta(n)$, but there is no p.r. function that calculates $\delta(n)$ for each $n$. 
Recursive Functions

- Recursive functions are defined just like p.r. ones with an additional clause.
- If $g$ is recursive and $\forall \vec{x} \exists z g(\vec{x}, z) = 0$ $f$ is defined as:

$$f(\vec{x}, \vec{y}) = \mu z g(\vec{x}, z) = 0$$

then $f$ is recursive.
- We can conclude that the property of being (the g.n. of a) recursive function is not itself recursive.
The recursive properties and relations

- The *characteristic function* of the numerical property $P$ is the one-place function $c_P$ such that:
  1. if $m$ is $P$, then $c_P(m) = 1$
  2. if $m$ isn’t $P$, then $c_P(m) = 0$. 
The p.r. properties and relations

- The characteristic function of the $k$-ary numerical relation $R$ is the $k$-place function $c_R$ such that

$$c_R(n_1 \ldots n_k) = \begin{cases} 1 & \text{if } n_1 \ldots n_k \text{ are } R \\ 0 & \text{otherwise} \end{cases}$$
Recursive properties and relations

- A *recursively decidable relation* is a relation with a recursive characteristic function.
- A *p.r. decidable relation* is a relation with a p.r. characteristic function.

It is not hard to argue that all recursive relations are decidable. The converse, which some have argued is not arguable at all, is called *Church’s Thesis.*
A non p.r. function

\[ A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0 \\
  A(m - 1, 1) & \text{if } n = 0 \\
  A(m - 1, A(m, n - 1)) & \text{otherwise}
\end{cases} \]

- This can be computed (by induction on \( \langle m, n \rangle \) lexicographically ordered).
- But this function is not primitive recursive.
- For each \( n \) there is a p.r. function that calculates \( A(n, n) \). But there is no p.r. function that calculates \( A(n, n) \) for each \( n \).