Theory of Computation

Course note based on *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd edition, authored by Martin Davis, Ron Sigal, and Elaine J. Weyuker.

course note prepared by

Tyng–Ruey Chuang

Week 3, Spring 2010

About This Course Note

- It is prepared for the course *Theory of Computation* taught at the National Taiwan University in Spring 2010.
- It follows very closely the book *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd edition, by Martin Davis, Ron Sigal, and Elaine J. Weyuker. Morgan Kaufmann Publishers. ISBN: 0-12-206382-1.
- It is available from Tyng-Ruey Chuang's web site:

  `http://www.iis.sinica.edu.tw/~trc/`

  and released under a Creative Commons "Attribution-ShareAlike 3.0 Taiwan" license:

  `http://creativecommons.org/licenses/by-sa/3.0/tw/`

1 Preliminaries (1)

1.1 Functions (1.2)

One-One Functions

- A function is *one-one* if, for all x, y in the domain of f, $f(x) = f(y)$ implies $x = y$.
- That is, if $x \neq y$, then $f(x) \neq f(y)$.
- Function $f(n) = n^2$ is one-one.
- Function $u_1^2(x_1, x_2) = x_1$ is not one-one as, for example, both $u_1^2(0, 0)$ and $u_1^2(0, 1)$ map to 0.

Onto Functions

- If the range of f is the set S, then we say f is an *onto* function with respect to S, or simply that f is *onto* S.
- Function $f(n) = n^2$ is onto the set of perfect squares $\{n^2 \mid n \in N\}$, but is not onto N.
- Let $S_1 \times S_2$ be domain of function $u_1^2(x_1, x_2) = x_1$, then function $u_1^2(x_1, x_2)$ is onto S_1.

2 Programs and Computable Functions (2)

2.1 Computable Functions (2.4)

Programs Accepting Any Number of Inputs

- We permit each program to be used with any number of inputs.
- If the program has n input variables, but only $m < n$ are specified, the remaining $n - m$ input variables are assigned the value 0 and the computation proceeds.
- On the other hand, if $m > n$ values are specified, then the extra input values are ignored.

Programs Accepting Any Number of Inputs, Examples

- Consider the following program $\mathscr{P}$ that computes $x_1 + x_2$,

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[B] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z \leftarrow Z - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{GOTO } B
\end{array}
$$

- We have

$$
\begin{aligned}
\Psi_{\mathscr{P}}^{(1)}(r_1) &= r_1 + 0 = r_1 \\
\Psi_{\mathscr{P}}^{(3)}(r_1, r_2, r_3) &= r_1 + r_2
\end{aligned}
$$

3 Primitive Recursive Functions (3)

3.1 PRC Classes (3.3)

Initial Functions

The following functions are called *initial functions*:

$$\begin{aligned} s(x) &= x+1, \\ n(x) &= 0, \\ u_i^n(x_1,\ldots,x_n) &= x_i, \quad 1 \le i \le n. \end{aligned}$$

Note: Function u_i^n is called the *projection function*. For example, $u_3^4(x_1, x_2, x_3, x_4) = x_3$.

Primitive Recursively Closed (PRC)

A class of total functions $\mathscr{C}$ is called a *PRC* class if

- the initial functions belong to $\mathscr{C}$,
- a function obtained from functions belonging to $\mathscr{C}$ by either composition or recursion also belongs to $\mathscr{C}$.

Computable Functions are Primitive Recursively Closed

Theorem 3.1. The class of computable functions is a PRC class. *Proof.* We have shown computable functions are closed under composition and recursion (Theorem 1.1 & 2.2). We need only verify the initial functions are computable. They are computed by the following programs.

$\boxed{s(x) = x+1}$ $Y \leftarrow X + 1$;

$\boxed{n(x)}$ the empty program;

$\boxed{u_i^n(x_1,\ldots,x_n)}$ $Y \leftarrow X_i$.

□

Primitive Recursive Functions

A function is called *primitive recursive* if it can be obtained from the initial functions by a finite number of applications of composition and recursion. Note that, by the above definition and the definition of Primitive Recursively Closed (PRC), it follows that: **Corollary 3.2.** The class of primitive recursive function is a PRC class.

Primitive Recursive Functions & PRC Classes

Theorem 3.3. A function is primitive recursive if and only if it belongs to every PRC class. *Proof.* ($\Leftarrow$) If a function belongs to every PRC class, then by Corollary 3.2, it belongs to the class of primitive recursive functions.

($\Rightarrow$) If f is primitive recursive, then there is a list of functions $f_1, f_2, \ldots, f_n$ such that $f_n = f$ and for each $f_i, 1 \leq i < n$, either

- f_i is an initial function, or
- f_i can be obtained from the preceding functions in the list by composition or recursion.

However, the initial functions belong to any PRC class $\mathscr{C}$. Furthermore, all functions obtained from functions in $\mathscr{C}$ by composition or recursion also belong to $\mathscr{C}$. It follows that each function $f_1, f_2, \ldots, f_n = f$ in the above list is in $\mathscr{C}$. □

Primitive Recursive Functions Are Computable

Corollary 3.4. Every primitive recursive function is computable. *Proof.* By Theorem 3.4, every primitive recursive function belongs to the PRC class of computable functions so is computable. □ Note that,

- If a function f is shown to be primitive recursive, by the above Corollary, f can be expressed as a program in language $\mathscr{S}$.
- Not only we know there is program in $\mathscr{S}$ for f, by Theorem 3.1 (1.1 & 2.2), we also know how to write this program.
- Furthermore, the program so written will always terminate.

However, if a function f is computable (that is, it is total and expressible in $\mathscr{S}$), it is not necessarily that f is primitive recursive. (A counter example will be shown later in this course.)

3.2 Some Primitive Recursive Functions/Predicates (3.4, 3.5)

Function $f(x, y) = x + y$ Is Primitive Recursive

Function f can be defined by the recursion equations:

$$\begin{aligned} f(x, 0) &= x, \\ f(x, y + 1) &= f(x, y) + 1. \end{aligned}$$

The above can be rewritten as

$$\begin{aligned} f(x, 0) &= u_1^1(x), \\ f(x, y + 1) &= g(y, f(x, y), x), \end{aligned}$$

where

$$g(x_1, x_2, x_3) = s(u_2^3(x_1, x_2, x_3)).$$

Function $h(x, y) = x \cdot y$ Is Primitive Recursive

Function h can be defined by the recursion equations:

$$\begin{aligned} h(x, 0) &= 0, \\ h(x, y + 1) &= h(x, y) + x. \end{aligned}$$

The above can be rewritten as

$$\begin{aligned} h(x, 0) &= n(x), \\ h(x, y + 1) &= g(y, h(x, y), x), \end{aligned}$$

where

$$\begin{aligned} g(x_1, x_2, x_3) &= f(u_2^3(x_1, x_2, x_3), u_3^3(x_1, x_2, x_3)), \\ f(x, y) &= x + y. \end{aligned}$$

Function $h(x) = x!$ Is Primitive Recursive

Function $h(x)$ can be defined by

$$\begin{aligned} h(0) &= 1, \\ h(t + 1) &= g(t, h(t)), \end{aligned}$$

where

$$g(x_1, x_2) = s(x_1) \cdot x_2.$$

Note that g is primitive recursive because

$$g(x_1, x_2) = s(u_1^2(x_1, x_2)) \cdot u_2^2(x_1, x_2).$$

Function *power*$(x, y) = x^y$ Is Primitive Recursive

Function *power* can be defined by

$$\begin{aligned} power(x, 0) &= 1, \\ power(x, y + 1) &= power(x, y) \cdot x. \end{aligned}$$

Note that these equations assign the value 1 to the "indeterminate" 0^0. The above definition can be further rewritten into ....

The Predecessor Function Is Primitive Recursive

The predecessor function *pred*(x) is defined as follows:

$$pred(x) = \begin{cases} x - 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

Note that function *pred* corresponds to the instruction $X \leftarrow X - 1$ in programming language $\mathscr{S}$. The above definition can be further rewritten into ....

Function $x \dot{-} y$ Is Primitive Recursive

Function $x \dot{-} y$ is defined as follows:

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y. \end{cases}$$

Note that function $x \dot{-} y$ is different from function $x - y$, which is undefined if $x < y$. In particular, $x \dot{-} y$ is total while $x - y$ is not. Function $x \dot{-} y$ is primitive recursive because

$$\begin{aligned} x \dot{-} 0 &= x, \\ x \dot{-} (t+1) &= \mathit{pred}(x \dot{-} t). \end{aligned}$$

The above definition can be further rewritten into ....

Function $|x - y|$ Is Primitive Recursive

Function $|x - y|$ can be defined as follows:

$$|x - y| = (x \dot{-} y) + (y \dot{-} x)$$

It is primitive recursive because the above definition can be further rewritten into ....

Is Function $\alpha(x)$ below Primitive Recursive?

Function $\alpha(x)$ is defined as:

$$\alpha(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x \neq 0. \end{cases}$$

It is primitive recursive because ....

$x = y$ Is Primitive Recursive

Is the function $d(x, y)$ below primitive recursive?

$$d(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

It is because $d(x, y) = \alpha(|x - y|)$.

Is $x \leq y$ Primitive Recursive?

It is primitive recursive because $x \leq y = \alpha(x \dot{-} y)$.

Logic Connectives Are Primitive Recursively Closed

Theorem 5.1. Let $\mathscr{C}$ be a PRC class. If P, Q are predicates that belong to $\mathscr{C}$, then so are $\sim P$, $P \vee Q$, and $P\&Q$. *Proof.* We define $\sim P$, $P \vee Q$, and $P\&Q$ as follows:

$$\begin{aligned} \sim P &= \alpha(P) \\ P \,\&\, Q &= P \cdot Q \\ P \vee Q &= \sim(\sim P \;\&\; \sim Q) \end{aligned}$$

We conclude that $\sim P$, $P \vee Q$, and $P\&Q$ all belong to $\mathscr{C}$. □

Logic Connectives Are Primitive Recursive and Computable

Corollary 5.2. If P, Q are primitive recursive predicates, then so are $\sim P$, $P \vee Q$, and $P\&Q$. **Corollary 5.3.** If P, Q are computable predicates, then so are $\sim P$, $P \vee Q$, and $P\&Q$.

Is $x < y$ Primitive Recursive?

It is primitive recursive because

$$x < y \;\Leftrightarrow\; \sim(y \leq x).$$

Definition by Cases

Theorem 5.4. Let $\mathscr{C}$ be a PRC class. Let functions g, h and predicate P belong to $\mathscr{C}$. Let function

$$f(x_1, \ldots, x_n) = \begin{cases} g(x_1, \ldots, x_n) & \text{if } P(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) & \text{otherwise.} \end{cases}$$

Then f belongs to $\mathscr{C}$. *Proof.* Function f belongs to $\mathscr{C}$ because

$$\begin{aligned} f(x_1, \ldots, x_n) &= g(x_1, \ldots, x_n) \cdot P(x_1, \ldots, x_n) \\ &+ h(x_1, \ldots, x_n) \cdot \alpha(P(x_1, \ldots, x_n)). \end{aligned}$$

□

Definition by Cases, More

Corollary 5.5. Let $\mathscr{C}$ be a PRC class. Let n-ary functions $g_1, \ldots, g_m, h$ and predicates $P_1, \ldots, P_m$ belong to $\mathscr{C}$, and let

$$P_i(x_1, \ldots, x_n) \;\&\; P_j(x_1, \ldots, x_n) \;=\; 0$$

for all $1 \le i \le j \le m$ and all $x_1, \ldots, x_n$. If

$$f(x_1, \ldots, x_n) = \begin{cases} g_1(x_1, \ldots, x_n) & \text{if } P_1(x_1, \ldots, x_n) \\ \vdots & \vdots \\ g_m(x_1, \ldots, x_n) & \text{if } P_m(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) & \text{otherwise.} \end{cases}$$

then f also belongs to $\mathscr{C}$. *Proof.* Proved by a mathematical induction on m. □

3.3 Iterated Operations and Bounded Quantifiers (3.6)

Iterated Operations

Theorem 6.1. Let $\mathscr{C}$ be a PRC class. If function $f(t, x_1, \ldots, x_n)$ belongs to $\mathscr{C}$, then so do the functions g and h

$$g(y, x_1, \ldots, x_n) = \sum_{t=0}^{y} f(t, x_1, \ldots, x_n)$$

$$h(y, x_1, \ldots, x_n) = \prod_{t=0}^{y} f(t, x_1, \ldots, x_n)$$

Proof. Functions g and h each can be recursively defined as

$$\begin{aligned} g(0, x_1, \ldots, x_n) &= f(0, x_1, \ldots, x_n), \\ g(t+1, x_1, \ldots, x_n) &= g(t, x_1, \ldots, x_n) + f(t+1, x_1, \ldots, x_n), \\ h(0, x_1, \ldots, x_n) &= f(0, x_1, \ldots, x_n), \\ h(t+1, x_1, \ldots, x_n) &= h(t, x_1, \ldots, x_n) \cdot f(t+1, x_1, \ldots, x_n). \end{aligned}$$

□

Iterated Operations, More

Corollary 6.2. Let $\mathscr{C}$ be a PRC class. If function $f(t, x_1, \ldots, x_n)$ belongs to $\mathscr{C}$, then so do the functions

$$g(y, x_1, \ldots, x_n) = \sum_{t=1}^{y} f(t, x_1, \ldots, x_n)$$

and

$$h(y, x_1, \ldots, x_n) = \prod_{t=1}^{y} f(t, x_1, \ldots, x_n).$$

In the above, we assume that

$$\begin{aligned} g(0, x_1, \ldots, x_n) &= 0, \\ h(0, x_1, \ldots, x_n) &= 1. \end{aligned}$$

Bounded Quantifiers

Theorem 6.3. If predicate $P(t, x_1, \ldots, x_n)$ belongs to some PRC class $\mathscr{C}$, then so do the predicates

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n)$$

and

$$(\exists t)_{\leq y} P(t, x_1, \ldots, x_n)$$

Proof. We need only observe that

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n) \;\Leftrightarrow\; \prod_{t=0}^{y} P(t, x_1, \ldots, x_n) = 1$$

and

$$(\exists t)_{\leq y} P(t, x_1, \ldots, x_n) \;\Leftrightarrow\; \sum_{t=0}^{y} P(t, x_1, \ldots, x_n) \neq 0$$

□

Bounded Quantifiers, More

Note that

$$(\forall t)_{<y} P(t, x_1, \ldots, x_n) \;\Leftrightarrow\; (\forall t)_{\leq y}[t = y \;\vee\; P(t, x_1, \ldots, x_n)],$$

and

$$(\exists t)_{<y} P(t, x_1, \ldots, x_n) \;\Leftrightarrow\; (\exists t)_{\leq y}[t \neq y \;\&\; P(t, x_1, \ldots, x_n)].$$

Therefore, both the quantifiers $(\forall t)_{<y}$ and $(\exists t)_{<y}$ are primitive recursively closed.

$y|x$ Is Primitive Recursive

The "y is a divisor of x" predicate $y|x$ is primitive recursive because

$$y|x \;\Leftrightarrow\; (\exists t)_{\leq x}(y \cdot t = x).$$

Prime(x) Is Primitive Recursive

The "x is a prime" predicate Prime(x) is primitive recursive because

$$\text{Prime}(x) \;\Leftrightarrow\; x > 1 \;\&\; (\forall t)_{\leq x}[t = 1 \;\vee\; t = x \;\vee\; \sim (t|x)].$$

3.4 Minimalization (3.7)

Bounded Minimalization

What does the following function g do?

$$g(y, x_1, \ldots, x_n) = \sum_{u=0}^{y} \prod_{t=0}^{u} \alpha(P(t, x_1, \ldots, x_n))$$

It computes the least value $t \leq y$ for which $P(t, x_1, \ldots, x_n)$ is true! To see why, let $t_0 \leq y$ such that

$$P(t, x_1, \ldots, x_n) = 0 \quad \text{for all } t < t_0,$$

but

$$P(t_0, x_1, \ldots, x_n) = 1$$

Then

$$\prod_{t=0}^{u} \alpha(P(t, x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } u < t_0, \\ 0 & \text{if } u \geq t_0. \end{cases}$$

Hence $g(y, x_1, \ldots, x_n) = \sum_{u<t_0} 1 = t_0$.

Bounded Minimalization, Continued

Define

$$\min_{t \leq y} P(t, x_1, \ldots, x_n) = \begin{cases} g(y, x_1, \ldots, x_n) & \text{if } (\exists t)_{\leq y} P(t, x_1, \ldots, x_n), \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\min_{t \leq y} P(t, x_1, \ldots, x_n)$, is the least value $t \leq y$ for which $P(t, x_1, \ldots, x_n)$ is true, if such exists; otherwise it assumes the (default) value 0. **Theorem 7.1.** $\min_{t \leq y} P(t, x_1, \ldots, x_n)$ is in PRC class $\mathscr{C}$ if $P(t, x_1, \ldots, x_n)$ is in $\mathscr{C}$. *Proof.* By Theorems 5.4 and 6.3. □

$\lfloor x/y \rfloor$ Is Primitive Recursive

$\lfloor x/y \rfloor$ is the "integer part" of the quotient x/y. The equation

$$\lfloor x/y \rfloor = \min_{t \leq x}[(t+1) \cdot y > x]$$

shows that $\lfloor x/y \rfloor$ is primitive recursive. Note that according to this definition, $\lfloor x/0 \rfloor = 0$.

$R(x, y)$, The Remainder Function, Is Primitive Recursive

$R(x, y)$ is the remainder when x is divided by y. As we can write

$$R(x, y) = x \dot{-} (y \cdot \lfloor x/y \rfloor)$$

so that $R(x, y)$ is primitive recursive. Note that $R(x, 0) = x$.

p_n, The nth Prime Number, Is Primitive Recursive

Note that $p_0 = 0, p_1 = 2, p_2 = 3, p_3 = 5$, etc. p_n is defined by the following recursive equations

$$\begin{aligned} p_0 &= 0, \\ p_{n+1} &= \min_{t \leq p_n! + 1} [\text{Prime}(t) \ \ \& \ \ t > p_n] \end{aligned}$$

so it is primitive recursive. Note that $p_n! + 1$ is not divisible by any of the primes $p_1, p_2, \ldots, p_n$. So, either $p_n! + 1$ is itself a prime or it is divisible by a prime greater than p_n. In either case, there is a prime q such that $p_n < q \leq p_n! + 1$.

p_n Is Primitive Recursive, Continued

To be precise, we shall first define a primitive recursive function

$$h(y, z) = \min_{t \leq z} [\text{Prime}(t) \ \ \& \ \ t > y].$$

Then we define another primitive function

$$k(x) = h(x, x! + 1)$$

Finally, p_n is defined as

$$\begin{aligned} p_0 &= 0, \\ p_{n+1} &= k(p_n), \end{aligned}$$

and it is concluded that p_n is primitive recursive.

Minimalization, With No Bound

We write

$$\min_y P(x_1, \ldots, x_n, y)$$

for the least value of y for which the predicate P is true *if there is one. If there is no value of y for which $P(x_1, \ldots, x_n, y)$ is true, then* $\min_y P(x_1, \ldots, x_n, y)$ *is undefined.* Note that unbounded minimalization of a predicate can easily produce function which is not total. For example,

$$x - y = \min_z \ [y + z = x]$$

is undefined for $x < y$.

Unbounded Minimalization is Partially Computable

Theorem 7.2. If $P(x_1, \ldots, x_n, y)$ is a computable predicate and if

$$g(x_1, \ldots, x_n) = \min_y P(x_1, \ldots, x_n, y)$$

then g is a partially computable function. *Proof.* The following program computes g:

[A] IF $P(X_1, \ldots, X_n, Y)$ GOTO E
$Y \leftarrow Y + 1$
GOTO A

□

3.5 Pairing Functions and Gödel Numbers (3.9)

Pairing Functions

- There is a one-one and onto function from $N \times N$ to N (with domain $N \times N$ and range N). This function is called a pairing function.
- That is, we can map a pair of numbers to a single number, and back, without losing information. Likewise, we can compute from any number a pair of numbers, and back, without missing anything.
- The primitive recursive function

$$\langle x, y \rangle = 2^x(2y + 1)\dot{-}1$$

  is a pairing function.
- $\langle 0, 0 \rangle = 0, \langle 1, 0 \rangle = 1, \langle 0, 1 \rangle = 2, \ldots$

The Pairing Function $\langle x, y \rangle = 2^x(2y + 1)\dot{-}1$

- Note that $2^x(2y + 1) \neq 0$, so

$$\langle x, y \rangle + 1 = 2^x(2y + 1)$$

- If z is any given number, then there is a *unique* solution x, y to the equation $\langle x, y \rangle = z$.
- Namely, x is the largest number such that $2^x | (z + 1)$, and y is then the solution of the equation $2y + 1 = (z + 1)/2^x$.
- The pairing function thus defines two functions l and r such that $x = l(z)$ and $y = r(z)$.

The Pairing Function $\langle x, y \rangle = 2^x(2y + 1)\dot{-}1$, Continued

If $\langle x, y \rangle = z$, then $x, y < z + 1$. Hence, $l(z) \leq z$, and $r(z) \leq z$. We can write

$$\begin{aligned} l(z) &= \min_{x \leq z}[(\exists y)_{\leq z}(z = \langle x, y \rangle)], \\ r(z) &= \min_{y \leq z}[(\exists x)_{\leq z}(z = \langle x, y \rangle)], \end{aligned}$$

so that $l(z)$ and $r(z)$ are primitive recursive functions.

Pairing Function Theorem

Theorem 8.1. The functions $\langle x, y\rangle, l(z)$, and $r(z)$ have the following properties:

1. they are primitive recursive;
2. $l(\langle x, y\rangle) = x, \quad r(\langle x, y\rangle) = y$;
3. $\langle l(z), r(z)\rangle = z$;
4. $l(z), r(z) \leq z$.

Gödel Number

We define the *Gödel Number* of the sequence $(a_1, \ldots, a_n)$ to be the number

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} p_i^{a_i}$$

Thus, the the Gödel number of the sequence (3, 1, 5, 4, 6) is

$$[3, 1, 5, 4, 6] = 2^3 \cdot 3^1 \cdot 5^5 \cdot 7^4 \cdot 11^6$$

For each fixed n, the function $[a_1, \ldots, a_n]$ is clearly primitive recursive. Note that the Gödel numbering method encodes and decodes arbitrary finite sequences of numbers.

Uniqueness Property of Gödel Numbering

Theorem 8.2. If $[a_1, \ldots, a_n] = [b_1, \ldots, b_n]$, then

$$a_i = b_i$$

for all $i = 1, \ldots, n$. □ This result is an immediate consequence of the uniqueness of the factorization of integers into primes, sometimes referred to as the *unique factorization theorem*. Note that,

$$1 = 2^0 = 2^0 3^0 = 2^0 3^0 5^0 = \ldots,$$

hence it is natural to regard 1 as the Gödel number of the "empty" sequence (i.e., the sequence of length 0).

Function $(x)_i$

We now define a primitive recursive function $(x)_i$ so that if

$$x = [a_1, \ldots, a_n]$$

then $(x)_i = a_i$. We set

$$(x)_i = \min_{t \leq x}(\sim p_i^{t+1} | x)$$

Note that $(x)_0 = 0$, and $(0)_i = 0$ for all i.

Function $Lt(x)$

We also define the "length" function Lt,

$$Lt(x) = \min_{i \leq x}[(x)_i \neq 0 \ \& \ (\forall j)_{\leq x}(j \leq i \ \vee \ (x)_j = 0)]$$

For example, if $x = 20 = 2^2 \cdot 5^1 = [2, 0, 1]$ then $(x)_1 = 2, (x)_2 = 0, (x)_3 = 1$, but $(x)_4 = 0, (x)_5 = 0, \ldots, (x)_i = 0$, for all $i \geq 4$. So $Lt(20) = 3$. Note that $Lt(0) = Lt(1) = 0$. If $x > 1$, and $Lt(x) = n$, then p_n divides x but no prime greater than p_n divides x.

Sequence Number Theorem

Theorem 8.3.

1.
$$([a_1, \ldots, a_n])_i = \begin{cases} a_i & \text{if } 1 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$$

2.
$$[(x)_1, \ldots, (x)_n] = x \ \text{ if } n \geq Lt(x).$$

□