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 7, 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/`

1 A Universal Program (4)

1.1 The Recursive Theorem (4.8)

Recursive Theorem

Theorem 8.1. Let $g(z, x_1, \ldots, x_m)$ be a partially computable function of $m+1$ variables. Then there is a number e such that

$$\Phi_e^{(m)}(x_1, \ldots, x_m) = g(e, x_1, \ldots, x_m)$$

Proof. Consider the partially computable function

$$g(S_m^1(v,v), x_1, \ldots, x_m)$$

where S_m^1 is the function that occurs in the parameter theorem. Then we have some number z_0 such that

$$\begin{aligned} g(S_m^1(v,v), x_1, \ldots, x_m) &= \Phi^{(m+1)}(x_1, \ldots, x_m, v, z_0) \\ &= \Phi^{(m)}(x_1, \ldots, x_m, S_m^1(v, z_0)). \end{aligned}$$

Setting $v = z_0$ and $e = S_m^1(z_0, z_0)$, we have

$$g(e, x_1, \ldots, x_m) = \Phi^{(m)}(x_1, \ldots, x_m, e) = \Phi_e^{(m)}(x_1, \ldots, x_m)$$

□

A Self-Reproducing Program

Corollary 8.2. There is a number e such that for all x

$$\Phi_e(x) = e$$

Proof. We consider the computable function

$$g(z, x) = u_1^2(z, x) = z$$

Applying the recursive theorem we obtain a number e such that

$$\Phi_e(x) = g(e, x) = e$$

□ Note: The program with number e "consumes' its input x and outputs a "copy" of itself. It is a "self-reproducing" organism!

Recursive Theorem, Examples

By using the recursive theorem, we can show that the functions obtained from primitive recursion over other computable functions are also computable. To see this, first consider

$$f(x,t) = \begin{cases} k & \text{if } t = 0 \\ g(t \dot{-} 1, \Phi_x(t \dot{-} 1)) & \text{otherwise} \end{cases}$$

where $g(x, y)$ is computable. By the recursion theorem there is a number e such that

$$\Phi_e(t) = f(e,t) = \begin{cases} k & \text{if } t = 0 \\ g(t \dot{-} 1, \Phi_e(t \dot{-} 1)) & \text{otherwise} \end{cases}$$

An induction on t shows that Φ_e is a total, and therefore computable, function. Now Φ_e satisfies the equations

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

That is, Φ_e is obtained from g by primitive recursion.

Fixed Point Theorem

Theorem 8.3. Let $f(z)$ be a computable function. Then there is a number e such that, for all x,

$$\Phi_{f(e)}(x) = \Phi_e(x)$$

Proof. Let $g(z, x) = \Phi_{f(z)}(x)$, a partially computable function. By the recursion theorem, there is a number e such that

$$\Phi_e(x) = g(e, x) = \Phi_{f(e)}(x)$$

□ Note that

- A number n is a fixed point of a function $f(x)$ if $f(n) = n$.
- However, there are computable functions that have no fixed point in this sense, e.g., $s(x)$.
- The fixed point theorem says that for every computable function $f(x)$, there is a number e of a program that *computes the same function* as the program with the number $f(e)$.

1.2 A Computable Function That is Not primitive Recursive (4.9)

A Computable Function That is Not primitive Recursive

The Plan for A Proof:

- Construct a computable function $\phi(t, x)$ that enumerates all of the unary primitive recursive functions. That is,
    1. for each fixed value $t = t_0$, the function $\phi(t_0, x)$ will be primitive recursive;
    2. for each unary primitive recursive function $f(x)$, there will be a number t_0 such that $f(x) = \phi(t_0, x)$.
- Show by diagonalization that the unary computable function $\phi(x, x) + 1$ is different from all primitive functions.
- Note that for the enumeration function $\phi(t, x)$ to work, we must show *all* primitive functions can be represented in an unary manner.

Reduce the Parameter Count in Primitive Recursion

From a total n-ary function f and a total $n + 2$-ary function g, one derives by primitive recursion a total $n + 1$-ary function h by

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

If $n > 1$ we can reduce the number of parameters needed from n to $n-1$ by using the pairing functions. That is, let

$$\begin{aligned}
\tilde{f}(x_1, \ldots, x_{n-1}) &= f(x_1, \ldots, x_{n-2}, l(x_{n-1}), r(x_{n-1})) \\
\tilde{g}(t, u, x_1, \ldots, x_{n-1}) &= g(t, u, x_1, \ldots, x_{n-2}, l(x_{n-1}), r(x_{n-1})) \\
\tilde{h}(x_1, \ldots, x_{n-1}, t) &= h(x_1, \ldots, x_{n-2}, l(x_{n-1}), r(x_{n-1}), t)
\end{aligned}$$

Reduce the Parameter Count in Primitive Recursion, Continued

Then we have

$$\begin{aligned}
\tilde{h}(x_1, \ldots, x_{n-1}, 0) &= \tilde{f}(x_1, \ldots, x_{n-1}) \\
\tilde{h}(x_1, \ldots, x_{n-1}, t+1) &= \tilde{g}(t, \tilde{h}(x_1, \ldots, x_{n-1}, t), x_1, \ldots, x_{n-1})
\end{aligned}$$

Note that the original function h can be retrieved by

$$h(x_1, \ldots, x_n, t) = \tilde{h}(x_1, \ldots, x_{n-2}, \langle x_{n-1}, x_n \rangle, t)$$

Primitive Recursion, Reduced Form

By iterating this process we can reduce the number of parameters to 1, that is, to recursions of the form

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

Recursions with no parameters can also be put in the above form. Namely, for recursion

$$\begin{aligned}
\psi(0) &= k \\
\psi(t+1) &= \theta(t, \psi(t))
\end{aligned}$$

we simply set

$$\begin{aligned}
f(x) &= k \\
g(x_1, x_2, x_3) &= \theta(u_1^3(x_1, x_2, x_3), u_2^3(x_1, x_2, x_3))
\end{aligned}$$

Then $\psi(t) = h(x, t)$ for all x.

Primitive Recursion, Further Reduced

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

The above can be further reduced by using the pairing function to combine arguments. Namely, we set

$$\tilde{h}(x,t) = \langle h(x,t), \langle x,t\rangle\rangle$$

Then, we have

$$\begin{aligned} \tilde{h}(x,0) &= \langle f(x), \langle x,0\rangle\rangle \\ \tilde{h}(x,t+1) &= \langle g(t,h(x,t),x), \langle x,t+1\rangle\rangle = \tilde{g}(\tilde{h}(x,t)) \end{aligned}$$

where

$$\tilde{g}(u) = \langle g(r(r(u)), l(u), l(r(u))), \langle l(r(u)), r(r(u))+1\rangle\rangle$$

Again, the original function h can be retrieved by $h(x,t) = l(\tilde{h}(x,t))$.

Taking Pairing Function as Initial Function

Theorem 9.1. The primitive recursive functions are precisely the functions obtainable from the initial functions

$$s(x), n(x), l(z), r(z), \langle x,y\rangle, \quad \text{and} \quad u_i^n, 1 \leq i \leq n$$

using the operations of composition and primitive recursion of the particular form

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

$\square$

Unary Primitive Recursive Function

Theorem 9.2. The unary primitive recursive functions are precisely those obtainable from the initial functions

$$s(x), n(x), l(z), r(z)$$

by applying the following three operations on unary functions:

1. to go from $f(x)$ and $g(x)$ to $f(g(x))$,
2. to go from $f(x)$ and $g(x)$ to $\langle f(x), g(x)\rangle$,
3. to go from $f(x)$ and $g(x)$ to the function defined by the recursion

$$\begin{aligned} h(0) &= 0 \\ h(t+1) &= \begin{cases} f(\frac{t}{2}) & \text{if } t+1 \text{ is odd,} \\ g(h(\frac{t+1}{2})) & \text{if } t+1 \text{ is even.} \end{cases} \end{aligned}$$

$\square$

Unary Primitive Recursive Function, Proof Outline

Proof Outline. Let **PR** be the set of all functions obtained from the initials listed in the theorem using operations 1 to 3. We show that **PR** is precisely the set of unary primitive recursive functions by proving the following:

1. show all functions in **PR** are primitive recursive,
2. show every unary primitive recursive function belongs to **PR**.

Because an unary primitive recursive function may be composed from primitive recursive functions that are not unary, e.g. $h(t)$ defined by $h'(t, \dots, t)$, where

$$h'(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

Proving 2. above will need additional care. $\square$

Functions in PR Are Primitive Recursive

We need only show that functions obtained from operation 3 are primitive recursive; the other cases are already known. Making use of Gödel numbering, we set

$$\begin{aligned} \vec{h}(0) &= 0, \\ \vec{h}(n) &= [h(0), \dots, h(n-1)] \ \text{ if } n > 0. \end{aligned}$$

We will show that $\vec{h}(n)$ is primitive recursive and then $h(n) = (\vec{h}(n+1))_{n+1}$ is primitive recursive as well. $\vec{h}(n)$ is primitive recursive because

$$\begin{aligned} \vec{h}(n+1) &= \vec{h}(n) \cdot p_{n+1}^{h(n)} \\ &= \begin{cases} \vec{h}(n) \cdot p_{n+1}^{f(\lfloor n/2 \rfloor)} & \text{if } n \text{ is odd,} \\ \vec{h}(n) \cdot p_{n+1}^{g((\vec{h}(n))_{\lfloor n/2 \rfloor})} & \text{if } n \text{ is even.} \end{cases} \end{aligned}$$

Recall that p_n is the n-th prime number.

Every Unary Primitive Recursive Function Is in PR, Proof Outline

- A function $g(x_1, \dots, x_n)$ is called *satisfactory* if it has the property that for any unary function $h_1(t), \dots, h_n(t)$ that belongs to **PR**, the unary function $g(h_1(t), \dots, h_n(t))$ also belongs to **PR**.
- Note that an unary function $g(t)$ that is satisfactory must belong to **PR** because $g(t) = g(u_1^1(t))$ and $u_1^1(t) = \langle l(t), r(t) \rangle$ belongs to **PR**.
- We proceed to show that all primitive recursive functions are satisfactory, hence prove that every unary primitive recursive function is in **PR**.
- We shall use the characterization of the primitive recursive functions of Theorem 9.1

All Primitive Recursive Functions Are Satisfactory, 1/3

- Initial functions: We need consider only the pairing function $\langle x_1, x_2 \rangle$ and the projection function u_i^n where $1 \leq i \leq n$.

  1. By definition, $\langle h_1(t), h_2(t) \rangle$ is in **PR** if both $h_1(t)$ and $h_2(t)$ are in **PR**.
  2. If $h_1(t), \ldots, h_n(t)$ are in **PR**, then $u_i^n(h_1(t), \ldots, h_n(t)) = h_i(t)$ of course is in **PR**.

- Function composition: Let

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

  where $g_1, \ldots, g_k$ and f are satisfactory. Let $h_1(t), \ldots, h_n(t)$ be given functions that belong to **PR**. Then, setting

$$\tilde{g}_i(t) = g_i(h_1(t), \ldots, h_n(t))$$

  for $1 \leq i \leq k$ we see that each $\tilde{g}_i$ is in **PR**. Now, the unary function

$$h(h_1(t), \ldots, h_n(t)) = f(\tilde{g}_1(t), \ldots, \tilde{g}_k(t))$$

  also belongs to **PR**, hence $h(x_1, \ldots, x_n)$ is satisfactory.

All Primitive Recursive Functions Are Satisfactory, 2/3

- Primitive recursion: Let

$$\begin{array}{rcl} h(x, 0) & = & f(x) \\ h(x, t+1) & = & g(h(x, t)) \end{array}$$

  where f and g are satisfactory. We want to encode the binary function $h(b, a)$ by an unary function $\psi(\langle a, b \rangle + 1) = h(b, a)$. Note that $\psi(0) = 0$ and $\psi(t+1) = h(r(t), l(t))$. Recall that

$$\langle a, b \rangle = 2^a(2b+1) - 1$$

  1. If $t+1$ is even, then $2^a(2b+1)$ is even; hence $a > 0$ and

$$\begin{array}{rcl} \psi(t+1) & = & h(b, a) = g(h(b, a-1)) \\ & = & g(\psi(2^{a-1}(2b+1))) = g(\psi((t+1)/2)). \end{array}$$

  2. If $t+1$ is odd, then $2^a(2b+1)$ is odd; hence $a = 0$ and

$$\psi(t+1) = h(b, 0) = f(b) = f(t/2).$$

All Primitive Recursive Functions Are Satisfactory, 3/3

- Primitive recursion (continued): In other words,

$$\begin{array}{rcl}\psi(0) & = & 0 \\ \psi(t+1) & = & \begin{cases} f(\frac{t}{2}) & \text{if } t+1 \text{ is odd,} \\ g(\psi(\frac{t+1}{2})) & \text{if } t+1 \text{ is even.}\end{cases}\end{array}$$

  Now f and g are satisfactory, and being unary, belongs to **PR**. By the definitions of **PR**, ψ belongs to **PR** as well.

- To retrieve h from ψ we simply use $h(b,a) = \psi(\langle a,b\rangle + 1)$. Therefore,

$$h(h_2(t), h_1(t)) = \psi(s(\langle h_1(t), h_2(t)\rangle))$$

  from which we see that if both h_1 and h_2 are in **PR** then so is $h(h_2(t), h_1(t))$. Hence h is satisfactory.

Enumerating All Unary Primitive Recursive Functions

We now define the function $\phi(t,x)$, also written as $\phi_t(x)$, to enumerate all unary primitive recursive functions:

$$\phi_t(x) = \begin{cases} x+1 & \text{if } t=0 \\ 0 & \text{if } t=1 \\ l(x) & \text{if } t=2 \\ r(x) & \text{if } t=3 \\ \phi_{l(n)}(\phi_{r(n)}(x)) & \text{if } t=3n+4, n\geq 0 \\ \langle\phi_{l(n)}(x), \phi_{r(n)}(x)\rangle & \text{if } t=3n+5, n\geq 0 \\ 0 & \text{if } t=3n+6, n\geq 0 \text{ and } x=0 \\ \phi_{l(x)}((x-1)/2) & \text{if } t=3n+6, n\geq 0 \text{ and } x \text{ is odd} \\ \phi_{r(x)}(\phi_t(x/2)) & \text{if } t=3n+6, n\geq 0 \text{ and } x \text{ is even} \end{cases}$$

A Closer Look at $\phi(t,x)$

- $\phi_0, \phi_1, \phi_2, \phi_3$ are the four initial functions.
- For $t > 3$, t is represented as $3n+i$ where $n \geq 0$ and $i = 4, 5, 6$. The three operations of Theorem 9.2 are then dealt with for the corresponding value of i.
- The pairing functions are used to guarantee all functions obtained for any value of t are eventually used in all possible applications of the three operations.
- It is clear from the definition that $\phi(t,x)$ is a total function and that it does enumerate all the unary primitive recursive functions.
- It is clear that the definition of $\phi(t,x)$ also provides an algorithm for computing the values of ϕ for any given inputs.

$\phi(t,x)$ **Is Computable**

We prove $\phi(t,x)$ is computable by using the recursive theorem. Let function $g(z,t,x)$ be defined as

$$g(z,t,x) =$$

$$\begin{cases} x+1 & \text{if } t=0 \\ 0 & \text{if } t=1 \\ l(x) & \text{if } t=2 \\ r(x) & \text{if } t=3 \\ \Phi_z^{(2)}(l(n), \Phi_z^{(2)}(r(n),x)) & \text{if } t=3n+4, n\geq 0 \\ \langle \Phi_z^{(2)}(l(n),x), \Phi_z^{(2)}(r(n),x)\rangle & \text{if } t=3n+5, n\geq 0 \\ 0 & \text{if } t=3n+6, n\geq 0 \text{ and } x=0 \\ \Phi_z^{(2)}(l(n), \lfloor x/2\rfloor) & \text{if } t=3n+6, n\geq 0 \text{ and } x \text{ is odd} \\ \Phi_z^{(2)}(r(n), \Phi_z^{(2)}(t, \lfloor x/2\rfloor) & \text{if } t=3n+6, n\geq 0 \text{ and } x \text{ is even} \end{cases}$$

$\phi(t,x)$ **Is Computable, Continued**

Then $g(z,t,x)$ is partially computable, and by the recursion theorem, there is a number e such that

$$g(e,t,x) = \Phi_e(t,x)$$

As $g(e,t,x)$ satisfy the definition of $\phi(t,x)$ and that definition determines ϕ uniquely as a total function, we must have

$$\phi(t,x) = g(e,t,x)$$

Hence, $\phi(t,x)$ is computable.

$\phi(x,x)+1$ **Is Not Primitive Recursive**

Theorem 9.3. The function $\phi(x,x)+1$ is a computable function that is not primitive recursive. □