CMPT 409: Theoretical Computer Science **Summer 2017**

Lecture 5: Computability (26 - 30 June)

Lecturer: Ternovska, Eugenia *Scribe: Lily Li*

5.1 Recursive Functions

Definition 5.1 *Minimization is another operations we need for recursion (not primitive recursion).*

$$f(\bar{x}) = \mu y[g(\bar{x}, y) = 0] \iff f(\bar{x}) \text{ is the least } b \text{ s.t. } g(\bar{x}, b) = 0 \text{ and } g(\bar{x}, i) \neq \infty \text{ for } i < b$$

$f(\bar{x}) = \infty$ *if no such* b *exists.*

Note: *here* 0 *represents true, while* 1 *represents false.*

If g *is computable then so is* f *since we can calculate* $g(\bar{x}, i)$ *for increasingly large* i *starting from* 0.

Definition 5.2 f *is recursive if and only* f *can be obtained from the initial functions from finitely many applications of primitive recursion, composition, and minimization.*

Observe that P.R. function is recursive, and all recursive function are computable (since we can apply induction on the number of primitive recursion, composition and minimizations used).

We want to show the converse (below), but first we need some background.

Theorem 5.3 *All computable functions (those computable by a RM) are recursive.*

In fact, any function computable in exponential time is P.R. More generally if f can be computed in time $T(x) = O(A_m(x))$ for some fixed m, where A_m is the Ackermann's function, then f is P.R.

Let us see some examples of how we define function using our basic tools of primitive recursion, complementation, and minimization (mostly just p.r.).

Example 5.4 *First we define the predecessor function* ρ*, taking care that the range is within* $\mathbb{N}$.

$$\rho(x) = \begin{cases} x - 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}$$

We can define ρ *by P.R. as* $\rho(0) = 0 = g$ *and* $\rho(x + 1) = x = h(x, \rho(x))$ *where* $g = Z$ *and* $h = I_{2,1}$.

Next we define the limited subtraction function Δ*:*

$$f_\Delta(x, y) = x \Delta y = \begin{cases} x - y & \text{if } y \leq x \\ 0 & \text{otherwise} \end{cases}$$

Define $f_\Delta(x, 0) = x = g(x)$ *where* $g(x) = I_{1,1}(x)$*. Further define* $f_\Delta(x, y+1) = h(x, y, x \Delta y) = \rho(x \Delta y)$ *where* ρ *is the predecessor function from before and* $h = \rho(I_{3,3})$.

Finally we define the maximum function using our limited subtraction in $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. *Define*

$$\max(x, y) = (x \Delta y) + y$$

Observe that if $x > y$ *then* $\max(x, y) = x$ *and if* $x \leq y$ *then* $\max(x, y) = y$ *as one would expect. Since both the subtraction and addition functions are P.R., their composition is P.R. as well.*

5.2 Relations and 0-1 Functions

For $R \subseteq \mathbb{N}^n$, n-ary relation. Define the characteristic function of R as:

$$R(\bar{x}) = \begin{cases} 0 & \text{if } \bar{x} \in R \\ 1 & \text{if } \bar{x} \notin R \end{cases}$$

Again note 0 is true and 1 is false for this part of the course. We consider R computable (resp. P.R.) if and only if the characteristic function is computable (resp. P.R.).

Next we will try to obtain logic using primitive recursive functions. But before we do that lets define a useful P.R. function sign, $\bar{sg} = 1 \Delta x$ i.e.

$$\bar{sg}(x) = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \end{cases}$$

Proposition 5.5 *If* R *and* S *are P.R. (resp. computable) then so are*

1. $\neg R$: $(\neg R)(\bar{x}) = \bar{sg}(R(x))$.
2. $R \vee S$: $(R \vee S)(\bar{x}) = R(\bar{x}) \cdot S(\bar{x})$.
3. $R \wedge S$: $(R \wedge S)(\bar{x}) = \neg(\neg R \vee \neg S)(\bar{x})$.

MOAR operations which preserves recursive functions and relations (as well as preserving computable relations and computable functions).

Example 5.6 ***Relations:***

$$(x < y) = \bar{sg}\,(y \Delta x) = \begin{cases} 0 & \textit{if } x < y \\ 1 & \textit{if } x \geq y \end{cases}$$
$$(x = y) = \neg(x < y) \wedge \neg(y < x)$$
$$(x \leq y) = (x < y) \vee (x = y)$$

Example 5.7 ***Bounded Sum***

$$g(\bar{x}, y) = \sum_{z<y} f(\bar{x}, z) = f(\bar{x}, 0) + \cdots + f(\bar{x}, y - 1)$$

g *can be defined from* f *by P.R. as:* $g(\bar{x}, 0) = 0$ *and* $g(\bar{x}, y + 1) = g(\bar{x}, y) + f(\bar{x}, y)$.

Example 5.8 ***Bounded Product***

$$h(\bar{x}, y) = \prod_{z<y} f(\bar{x}, z) = f(\bar{x}, 0) \cdots f(\bar{x}, y-1).$$

Similarly, h can be defined from f by P.R. as $h(\bar{x}, 0) = 1 = g$ and $h(\bar{x}, y+1) = h(\bar{x}, y) \cdot f(\bar{x}, y)$.

Example 5.9 ***Bounded Quantification***

$$\begin{aligned} S(\bar{x}, y) &= \exists z < y R(\bar{x}, z) \\ &= \prod_{z<y} R(\bar{x}, z) = R(\bar{x}, 0) \cdots R(\bar{x}, y-1) \\ &= h(\bar{x}, y) \end{aligned}$$

Similarly, the universal quantifier is P.R. since

$$\begin{aligned} T(\bar{x}, y) &= \forall z < y R(\bar{x}, z) \\ &= \neg \exists z < y \neg R(\bar{x}, z) \end{aligned}$$

Thus the computability of R implies the computability of S and T.

Example 5.10 ***Number Theoretical Functions***

$$\begin{aligned} \mathsf{Prime}(x) &= \Big\{ 0 \text{ if } x \text{ is prime } 1 \text{ otherwise} \\ &= (1 < x) \wedge (\forall z < x : \neg(z|x) \vee z = 1) \end{aligned}$$

where

$$x|y = \exists z \leq y : x \cdot z = y$$

Definition 5.11 *We will define a conditional by cases:*

$$\begin{aligned} f(\bar{x}) &= \begin{cases} g(\bar{x}) \text{ if } R(\bar{x}) \\ h(\bar{x}) \text{ otherwise} \end{cases} \\ &= Cond(R(\bar{x}), g(\bar{x}), h(\bar{x})) \\ &= \bar{sg}(x) \cdot y + \bar{sg}(\bar{sg}(x)) \cdot z \end{aligned}$$

Where

$$Cond(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{if } x > 0 \end{cases}$$

As before, if g, h, R are P.R. (resp. computable) then so is f.

Example 5.12 ***Bounded Minimization***

$$f(\bar{x}, y) = \min z < y R(\bar{x}, z) = \begin{cases} \text{least } z \text{ s.t. } z < y \text{ and } f & \text{if such } z \text{ exists} \\ y & \text{otherwise} \end{cases}$$

Notice that (1) f is always total since R is a total $0-1$ valued function, and (2) f is P.R. (resp. computable) whenever R is P.R. (resp. computable) since

$$f(\bar{x}, y) = \sum_{z<y} (\exists v \leq z : R(\bar{x}, v))$$

The idea is to use bounded summation as bounded minimization.

5.3 Simulating RM with Recursive Function

We want to consider all computable function (those computable by RM) and show that these can be expressed recursively. Consider

$$T([P], \bar{x}, y) \iff y \text{ is a halting computation of program } P \text{ on input } \bar{x}$$

where $[P]$ is the encoding of program P. If T is P.R. then the function $\min_y : T([P], \bar{x}, y)$ is recursive and so the function associated with P is recursive. In the following we construct T and show that it is P.R.

5.4 Gödel Numbering

We wish to encode the follow: commands of RMs, programs (sequences of commands), states, computations (sequences of states) as natural numbers. To do this, we must first define the i^{th} prime function $f(i) = p_i$ and note that $f(i)$ is P.R. Let

$$\begin{aligned} p_0 &= f(0) & &= 2 \\ p_{x+1} &= f(x+1) & &= \min y < p_x^{p_x} \end{aligned}$$

This means that $p_x < y$ and $\mathsf{Prime}(y)$ so $f(x+1)$ is the first prime after p_x.

Next we need to observe that prime decomposition is also P.R. Notation: z_x is the exponent of p_x in the prime decomposition of z where

$$z = p_0^{z_0} \cdot p_1^{z_1} \cdots p_m^{z_m}$$

Thus we can write z_x as

$$\min y < z : \neg(p_x^{y+1} | z)$$

Finally we need length $l(z)$ to be P.R. Length is defined as follows:

$$l(z) = m + 1 \qquad \text{where } z = p_0^{z_0} \cdots p_m^{z_m}.$$

Alternatively, we can encode $l(z)$ as

$$\begin{aligned} l(z) &= \min y < z : \prod_{x<y} P_x^{z_x} = z \\ &= 1 + \text{ subscript of largets prime which divides } z \\ &= \max y < z : (p_y | z) + 1 \end{aligned}$$

5.4.1 Numbering Programs

Using the three P.R functions above, we can number programs as follows. If $P = \langle c_0, ..., c_{n-1} \rangle$ then $\#(P) = p_0^{\#(c_0)} \cdots p_m^{\#(c_{n-1})}$ where

$$\begin{aligned} \#(z_i) &= 2^i \\ \#(s_i) &= 3^i \\ \#(J_{i,j,k}) &= 5^i 7^j 11^k \end{aligned}$$

Thus distinct programs get distinct numbers and the encoding $\#P$ of P is P.R.

5.4.2 Encoding Relations

$$\begin{aligned}
\mathsf{Succ}(x,i) &\iff x = \#(s_i) \\
\mathsf{Zero}(x,i) &\iff x = \#(z_i) \\
\mathsf{Jump}(x,i,j,k) &\iff x = \#(J_{i,j,k}) \\
\mathsf{Command}(x) &\iff \exists i < x : (\mathsf{Succ}(x,i) \vee \mathsf{Zero}(x,i)) \vee (\exists i < x, \exists j < x, \exists k < x : \mathsf{Jump}(x,i,j,k))
\end{aligned}$$

Let $\mathsf{Prog}(z) \iff z = \#(P)$ for some program P. $\mathsf{Prog}(z)$ is P.R since

$$\begin{aligned}
\mathsf{Prog}(z) &\iff z \text{ encodes a sequence of commands} \\
&\iff \forall j < l(z) : \mathsf{Command}(z_j)
\end{aligned}$$

5.4.3 States

Consider RM program P, which uses registers $R_1, ..., R_m$. A state for P is $s = \langle k, R_1, ..., R_m \rangle$ where k is the index of the next command to execute and $R_1, ..., R_m$ is the value in each register.

Next we consider computations. Define

$$\hat{z} = \begin{cases} z \text{ if } Prog(z) \text{ encodes a program} \\ 1 \text{ otherwise} \end{cases}$$

And for our use define

$$\{z\} = \{$$

$\mathsf{Nex}(u,z)$ = the code of the state u' which results when program $\{z\}$ executes one step from the state encoded by u. If u is a halting state $u' = u$. We need to show that Nex is P.R.

5.5 Kleene T Predicate

Recall $T(P, \bar{x}, y)$ from above. Using Gödel encoding we can rewrite T as $T_n(z, x_1, ..., x_n, y) = y$ encodes the computation of $\{z\}$ on input $\bar{x} = (x_1, ..., x_n)$ where T_n is a $n+2$-ary relation.