Universal Program*

Xiaofeng Gao

Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R.China

CS363-Computability Theory

* Special thanks is given to Prof. Yuxi Fu for sharing his teaching materials.

Outline

Outline

General Remark

There are universal programs that embody all the programs.

A program is universal if upon receiving the Gödel number of a program it simulates the program indexed by the number.

Intuition

Consider the function $\psi(x,y)$ defined as follows

$$\psi(x,y) \simeq \phi_x(y).$$

In an obvious sense $\psi(x,\_)$ is a universal function for the unary funcitons

$$\phi_0, \phi_1, \phi_2, \phi_3, \ldots.$$

Universal Function

The universal function for n-ary computable functions is the $(n+1)$-ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write ψ_U for $\psi_U^{(1)}$.

Universal Function

The universal function for n-ary computable functions is the $(n+1)$-ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write ψ_U for $\psi_U^{(1)}$.

Question: Is $\psi_U^{(n)}$ computable?

The Theorem

Theorem. For each n, the universal function $\psi_U^{(n)}$ is computable.

The Theorem

Theorem. For each n, the universal function $\psi_U^{(n)}$ is computable.

Proof. Given a number e, decode the number to get the program P_e; and then simulate the program P_e. If the simulation ever terminates, then return the number in R_1. By Church-Turing Thesis, $\psi_U^{(n)}$ is computable.

Proof in Detail

The states of the computation of the program $P_e(\mathbf{x})$ can be described by a configuration and an instruction number.

Proof in Detail

The states of the computation of the program $P_e(\mathbf{x})$ can be described by a configuration and an instruction number.

A state can be coded up by the number

$$\sigma = \pi(c, j),$$

where c is the configuration that codes up the current values in the registers

$$c = 2^{r_1} 3^{r_2} \ldots = \prod_{i \geq 1} p_i^{r_i},$$

and j is the next instruction number.

Step 1: Three New $(n+2)$-ary functions

- Define two new functions c_n and j_n:

$$\begin{aligned} \mathsf{c}_n(e,\mathbf{x},t) &= \text{the configuration after } t \text{ steps of } P_e(\mathbf{x}), \\ \mathsf{j}_n(e,\mathbf{x},t) &= \text{the number of the next instruction after } t \text{ steps} \\ &\quad\ \text{of } P_e(\mathbf{x}) \text{ (it is 0 if } P_e(\mathbf{x}) \text{ stops in } t \text{ or less steps)}, \end{aligned}$$

- If the computation of $P_e(\mathbf{x})$ stops, it does so in $\mu t(\mathsf{j}_n(e,\mathbf{x},t)=0)$ steps, and the final configuration is $\mathsf{c}_n(e,\mathbf{x},\mu t(\mathsf{j}_n(e,\mathbf{x},t)=0))$.

$$\psi_U^{(n)}(e,\mathbf{x}) \simeq (\mathsf{c}_n(e,\mathbf{x},\mu t(\mathsf{j}_n(e,\mathbf{x},t)=0)))_1$$

- Let $\sigma_n(e,\mathbf{x},t)=\pi(\mathsf{c}_n(e,\mathbf{x},t),\mathsf{j}_n(e,\mathbf{x},t))$. If σ_n is primitive recursive, then $\mathsf{c}_n,\mathsf{j}_n$ are primitive recursive!

Step 2: Computability of $\sigma_n(e, \mathbf{x}, t)$

The function σ_n can be defined by recursion as follows:

$$\begin{aligned}\sigma_n(e, \mathbf{x}, 0) &= \pi(2^{x_1}3^{x_2}\dots p_n^{x_n}, 1),\\ \sigma_n(e, \mathbf{x}, t+1) &= \pi(\mathsf{config}(e, \sigma_n(e, \mathbf{x}, t)), \mathsf{next}(e, \sigma_n(e, \mathbf{x}, t))),\end{aligned}$$

$$\mathsf{config}(e, \pi(c, j)) = \begin{cases} \text{New configuration after} & \text{if } 1 \le j \le s \\ j^{th} \text{ instruction of } P_e \text{ is obeyed,} & \\ & \\ c, & \text{otherwise.}\end{cases}$$

$$\mathsf{next}(e, \pi(c, j)) = \begin{cases} \text{No. of next instruction after} & \text{if } 1 \le j \le s \\ j^{th} \text{ instruction of } P_e \text{ is obeyed on } c, & \text{and it exists} \\ & \\ 0, & \text{otherwise.}\end{cases}$$

If config and next are primitive recursive, then so is σ_n!

Step 3: Computability of config and next

$$\mathsf{ln}(e) = \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) = \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \leq j \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathsf{ch}(c,z) = \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

$$\mathsf{v}(c,j,z) = \begin{cases} \text{the number } j' \text{ of the next instruction when the configuration } c \text{ is operated on by the } j\text{th instruction with code } z, & \text{if } j > 0, \\ 0, & \text{if } j = 0. \end{cases}$$

Step 3: Computability of config and next (2)

We can define the function $\mathsf{config}(\_,\_)$ by

$$\mathsf{config}(e,\sigma) = \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \le \pi_2(\sigma) \le \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

and the function $\mathsf{next}(\_,\_)$ by

$$\mathsf{next}(e,\sigma) = \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \le \pi_2(\sigma) \le \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

If ln, gn, ch, and v are primitive recursive, then so are config and next!

Step 4: Computability of ln, gn, ch, and v

Any number $x \in \mathbb{N}$ has a unique expression as

(a) $x = \sum_{i=0}^{\infty} \alpha_i 2^i$, with $\alpha_i = 0$ or 1, all i.

(b) $x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_l}$, with $0 \leq b_1 < b_2 < \ldots < b_l$ and $l \geq 1$.

(c) $x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_1+a_2+\ldots+a_k+k-1}$.

Define α, ℓ, b, and a as follows:

$$\alpha(i,x) = \alpha_i \text{ as in the expression (a);}$$

$$\ell(x) = \begin{cases} \ell \text{ as in (b)}, & \text{if } x > 0, \\ 0 & \text{otherwise;} \end{cases}$$

$$b(x) = \begin{cases} b_i \text{ as in (b)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

$$a(i,x) = \begin{cases} a_i \text{ as in (c)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

Each of the functions α, ℓ, b, a is computable.

ln and gn are primitive recursive

Both functions are primitive recursive since

$$\begin{aligned} \mathsf{ln}(e) &= \ell(e+1), \\ \mathsf{gn}(e,j) &= \mathsf{a}(j, e+1). \end{aligned}$$

Computability of ch, and v

Define primitive recursive functions u, u_1, u_2, v_1, v_2, and v_3:

$\mathsf{u}(z) = m$ whenever $z = \beta(Z(m))$ or $z = \beta(S(m))$:

$$\mathsf{u}(z) = \mathsf{qt}(4, z) + 1.$$

$\mathsf{u}_1(z) = m_1$ and $\mathsf{u}_2(z) = m_2$ whenever $z = \beta(T(m_1, m_2))$:

$$\begin{aligned} \mathsf{u}_1(z) &= \pi_1(\mathsf{qt}(4, z)) + 1, \\ \mathsf{u}_2(z) &= \pi_2(\mathsf{qt}(4, z)) + 1. \end{aligned}$$

$\mathsf{v}_1(z) = m_1$ and $\mathsf{v}_2(z) = m_2$ and $\mathsf{v}_3(z) = q$ if $z = \beta(J(m_1, m_2, q))$:

$$\begin{aligned} \mathsf{v}_1(z) &= \pi_1(\pi_1(\mathsf{qt}(4, z))) + 1, \\ \mathsf{v}_2(z) &= \pi_2(\pi_1(\mathsf{qt}(4, z))) + 1, \\ \mathsf{v}_3(z) &= \pi_2(\mathsf{qt}(4, z)) + 1. \end{aligned}$$

Computability of ch, and v

Define primitive recursive functions zero, succ, and rans:

The change in the configuration c effected by instruction $Z(m)$:

$$\mathsf{zero}(c, m) = \mathsf{qt}(p_m^{(c)_m}, c).$$

The change in the configuration c effected by instruction $S(m)$:

$$\mathsf{succ}(c, m) = p_m c.$$

The change in the configuration c effected by instruction $T(m, n)$:

$$\mathsf{tran}(c, m, n) = \mathsf{qt}(p_n^{(c)_n}, p_n^{(c)_m} c).$$

ch, and v are primitive recursive

$$\mathsf{ch}(c,z) = \begin{cases} \mathsf{zero}(c,\mathsf{u}(z)), & \text{if } \mathsf{rm}(4,z) = 0, \\ \mathsf{succ}(c,\mathsf{u}(z)), & \text{if } \mathsf{rm}(4,z) = 1, \\ \mathsf{tran}(c,\mathsf{u}_1(z),\mathsf{u}_2(z)), & \text{if } \mathsf{rm}(4,z) = 2, \\ c, & \text{if } \mathsf{rm}(4,z) = 3. \end{cases}$$

$$\mathsf{v}(c,j,z) = \begin{cases} j+1, & \text{if } \mathsf{rm}(4,z) \neq 3, \\ j+1, & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} \neq (c)_{\mathsf{v}_2(z)}, \\ \mathsf{v}_3(z), & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} = (c)_{\mathsf{v}_2(z)}. \end{cases}$$

Conclusion

We conclude that the functions $\mathsf{c}_n, \mathsf{j}_n, \sigma_n$ are primitive recursive.

Step 1		Step 2		Step 3		Step 4
$\mathsf{c}_n(e,\widetilde{x},t)$ $\mathsf{j}_n(e,\widetilde{x},t)$ $\sigma_n(e,\widetilde{x},t)$	$\Rightarrow$	$\mathsf{config}(e,\pi(c,j))$ $\mathsf{next}(e,\pi(c,j))$	$\Rightarrow$	$\mathsf{ln}(e)$ $\mathsf{gn}(e,j)$ $\mathsf{ch}(c,z)$ $\mathsf{v}(c,j,z)$	$\Rightarrow$	$\alpha(i,x)$, $\ell(x)$, $b(i,x)$, $a(i,x)$ u, u_1, u_2, v_1, v_2, and v_3 $\mathsf{zero}(c,m)$, $\mathsf{succ}(c,m)$, $\mathsf{tran}(c,m,n)$

Further Constructions

For each $n \geq 1$, the following predicates are primitive recursive:

1. $\mathsf{S}_n(e, \mathbf{x}, y, t) \stackrel{\text{def}}{=}$ '$P_e(\mathbf{x}) \downarrow y$ in t or fewer steps'.
2. $\mathsf{H}_n(e, \mathbf{x}, t) \stackrel{\text{def}}{=}$ '$P_e(\mathbf{x}) \downarrow$ in t or fewer steps'.

Further Constructions

For each $n \geq 1$, the following predicates are primitive recursive:

1. $\mathsf{S}_n(e, \mathbf{x}, y, t) \stackrel{\text{def}}{=}$ ‘$P_e(\mathbf{x}) \downarrow y$ in t or fewer steps’.
2. $\mathsf{H}_n(e, \mathbf{x}, t) \stackrel{\text{def}}{=}$ ‘$P_e(\mathbf{x}) \downarrow$ in t or fewer steps’.

They are defined by

$$
\begin{aligned}
\mathsf{S}_n(e, \mathbf{x}, y, t) &\stackrel{\text{def}}{=} \mathsf{j}_n(e, \mathbf{x}, t) = 0 \wedge (\mathsf{c}_n(e, \mathbf{x}, t))_1 = y, \\
\mathsf{H}_n(e, \mathbf{x}, t) &\stackrel{\text{def}}{=} \mathsf{j}_n(e, \mathbf{x}, t) = 0.
\end{aligned}
$$

Kleene's Normal Form Theorem

Theorem. (Kleene)
There is a primitive recursive function $\mathsf{U}(x)$ and for each $n \geq 1$ a primitive recursive predicate $\mathsf{T}_n(e, \mathbf{x}, z)$ such that

1. $\phi_e^{(n)}(\mathbf{x})$ is defined if and only if $\exists z.\mathsf{T}_n(e, \mathbf{x}, z)$.
2. $\phi_e^{(n)}(\mathbf{x}) \simeq \mathsf{U}(\mu z T_n(e, \mathbf{x}, z))$.

Kleene's Normal Form Theorem

Theorem. (Kleene)
There is a primitive recursive function $\mathsf{U}(x)$ and for each $n \geq 1$ a primitive recursive predicate $\mathsf{T}_n(e, \mathbf{x}, z)$ such that

1. $\phi_e^{(n)}(\mathbf{x})$ is defined if and only if $\exists z.\mathsf{T}_n(e, \mathbf{x}, z)$.
2. $\phi_e^{(n)}(\mathbf{x}) \simeq \mathsf{U}(\mu z T_n(e, \mathbf{x}, z))$.

Proof. Let $\mathsf{T}_n(e, \mathbf{x}, z) = S_n(e, \mathbf{x}, (z)_1, (z)_2)$. Then (1) is clear. For (2) let $\mathsf{U}(x) = (x)_1$. Then

$$\phi_e^{(n)}(\mathbf{x}) \simeq \mathsf{U}(\mu z.\mathsf{T}_n(e, \mathbf{x}, z)).$$

Every computable function can be obtained from a primitive recursive function by using at most one application of the μ-operator in a standard manner.

Outline

Application: Undecidability

Theorem. The problem 'ϕ_x is total' is undecidable.

Application: Undecidability

Theorem. The problem 'ϕ_x is total' is undecidable.

Proof. If 'ϕ_x is total' were decidable, then by Church's Thesis

$$f(x) = \begin{cases} \psi_U(x,x)+1, & \text{if } \phi_x \text{ is total}, \\ 0, & \text{if } \phi_x \text{ is not total}. \end{cases}$$

would be a total computable function that differs from every total computable function.

Application: Nonprimitive Total Computable Function

Theorem. There is a total computable function that is not primitive recursive.

Application: Nonprimitive Total Computable Function

Theorem. There is a total computable function that is not primitive recursive.

Proof.
1. The primitive recursive functions are effectively denumerable.

2. Construct a coding of a primitive recursive function $f(x)$ one can effectively calculate $p(e)$ such that $\phi_{p(e)}(x) \simeq f(x)$.

3. But then $g(x) = \phi_{p(x)}(x) + 1 = \psi_U(p(x), x) + 1$ is a total computable function that is not primitive recursive.

Proof (1)

$Sub(f; g_1, g_2, \cdots, g_m)$ denotes the function obtained by substituting $g_1, \cdots, g_m$ into f. (f is m-ary; g_i are n-ary for some n).

$Rec(f, g)$ denotes the function obtained from f and g by recursion (f is n-ary, g is $(n+2)$-ary for some n).

S denotes the function $x + 1$

U_i^n denotes the projection function $U_i^n(x_1, \cdots, x_n) = x_i$.

For each primitive recursive function, we have a Plan to indicate the basic functions used and the exact sequence of operations performed.

Example: $f(x) = x^2$

$g_1 = Sub(S; U_3^3)$: $\quad g_1(x,y,z) = U_3^3(x,y,z) + 1 = z + 1$

$g_2 = Rec(U_1^1; g_1)$: $\quad \begin{cases} g_2(x,0) = U_1^1(x) = x, \\ g_2(x,y+1) = g_1(x,y,g_2(x,y)) = g_2(x,y) + 1 \end{cases}$

So $g_2(x,y) = x + y$

$g_3 = Sub(g_2; U_1^3, U_3^3)$: $\quad g_3(x,y,z) = g_2(x,z) = x + z$

$g_4 = Rec(0; g_3)$: $\quad \begin{cases} g_4(x,0) = 0, \\ g_4(x,y+1) = g_3(x,y,g_4(x,y)) = x + g_4(x,y) \end{cases}$

So $g_4(x,y) = xy$

$f = Sub(g_4; U_1^1, U_1^1)$: $\quad f(x) = g_4(x,x) = x^2$

Effective Numbering

Now restrict our attention to plans for unary primitive recursive functions. We can number these plans in an effective way. Define:

$$\theta_n \quad = \quad \text{the unary primitive recursive function defined by plan number } n$$

Since every primitive recursive function is computable, there is a total function p such that for each n, $p(n)$ is the number of a program that computes θ_n.

$$\theta_n = \phi_{p(n)}.$$

Computability of $p(n)$

We know how to obtain a program for the function $Sub(f; g_1, \cdots, g_m)$ given programs for $f, g_1, \cdots, g_m$;

We know how to obtain a program for the function $Rec(f, g)$ given programs for f, g;

We have explicit programs for the basic functions.

Hence, given a plan for a primitive recursive function f involving intermediate functions $g_1, \cdots, g_k$, we can effectively find programs for $g_1, \cdots, g_k$ and finally f.

Thus, by Church's Thesis, there is an effectively computable function p such that $\theta_n = \phi_{p(n)}$.

Construction of Total Non-Primitive Recursive Function

For every primitive recursive function θ_n, we use a diagonal construction as follows:

$$\begin{aligned} g(x) &= \theta_x(x) + 1 \\ &= \phi_{p(x)}(x) + 1 \\ &= \psi_U(p(x), x) + 1 \end{aligned}$$

g is a total function that is not primitive recursive, but g is computable, by the computability of ψ_U and p.

Outline

1. Universal Functions and Universal Programs
2. Application of the Universal Program
3. Effective Operations on Computable Functions

Application: Effectiveness of Function Operation

Fact. There is a total computable function $s(x, y)$ such that $\phi_{s(x,y)} = \phi_x \phi_y$ for all x, y.

Application: Effectiveness of Function Operation

Fact. There is a total computable function $s(x,y)$ such that $\phi_{s(x,y)} = \phi_x\phi_y$ for all x, y.

Proof. Let $f(x,y,z) = \phi_x(z)\phi_y(z) = \psi_U(x,z)\psi_U(y,z)$.
By S-m-n Theorem there is a total function $s(x,y)$ such that $\phi_{s(x,y)}(z) \simeq f(x,y,z)$.

Application: Effectiveness of Set Operation

Fact. There is a total computable function $s(x, y)$ such that $W_{s(x,y)} = W_x \cup W_y$.

Application: Effectiveness of Set Operation

Fact. There is a total computable function $s(x, y)$ such that $W_{s(x,y)} = W_x \cup W_y$.

Proof. Let

$$f(x, y, z) = \begin{cases} 1, & \text{if } z \in W_x \text{ or } z \in W_y, \\ \text{undefined}, & \text{otherwise}. \end{cases}$$

By S-m-n Theorem there is a total function $s(x, y)$ such that $\phi_{s(x,y)}(z) \simeq f(x, y, z)$. Clearly $W_{s(x,y)} = W_x \cup W_y$.

Application: Effectiveness of Inversion

Let $g(x,y)$ be a computable function such that
(a) $g(x,y)$ is defined iff $y \in E_x$;
(b) If $y \in E_x$, then $g(x,y) \in W_x$ and $\phi_x(g(x,y)) = y$. (i.e., $g(x,y) \in \phi_x^{-1}(\{y\})$)

Application: Effectiveness of Inversion

Let $g(x,y)$ be a computable function such that
(a) $g(x,y)$ is defined iff $y \in E_x$;
(b) If $y \in E_x$, then $g(x,y) \in W_x$ and $\phi_x(g(x,y)) = y$. (i.e., $g(x,y) \in \phi_x^{-1}(\{y\})$)

By S-m-n Theorem, there is a total computable function k such that $g(x,y) \simeq \phi_{k(x)}(y)$. Then from (a) and (b) we have:
(a') $W_{k(x)} = E_x$;
(b') $E_{k(x)} \subseteq W_x$; If $y \in E_x$, then $\phi_x(\phi_{k(x)}(y)) = y$.

Hence if ϕ_x is injective, then $\phi_{k(x)} = \phi_x^{-1}$ and $E_{k(x)} = W_x$.

Application: Effectiveness of Recursion

Consider f defined by the following recursion

$$f(e_1, e_2, \mathbf{x}, 0) \simeq \phi_{e_1}^{(n)}(\mathbf{x}) \simeq \psi_U^{(n)}(e_1, \mathbf{x}),$$

and

$$\begin{aligned} f(e_1, e_2, \mathbf{x}, y+1) &\simeq \phi_{e_2}^{(n+2)}(\mathbf{x}, y, f(e_1, e_2, \mathbf{x}, y)) \\ &\simeq \psi_U^{(n+2)}(e_2, \mathbf{x}, y, f(e_1, e_2, \mathbf{x}, y)). \end{aligned}$$

By S-m-n Theorem, there is a total computable function $r(e_1, e_2)$ such that

$$\phi_{r(e_1,e_2)}^{(n+1)}(\mathbf{x}, y) \simeq f(e_1, e_2, \mathbf{x}, y).$$