CS 260
Design and Analysis of Algorithms
10. Computations and Unsolvable Problems

Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

Model of Computation

To understand more deeply the notions of algorithm and computable function we should fix an universal model of computation. Turing machine is considered often as such a model.

However, we will not study Turing machines but will consider programs in an universal programming language which compute functions from ω^t to ω, $t \geq 1$. It may be Fortran or C^{++}, for example.

Model of Computation

Such programs implement exactly the class of partial recursive functions. Of course, we must assume that programs have enough time and enough memory (potentially infinite memory which can be increased).

Enumeration of Programs

The set of all considered programs is a denumerable set $P_0, P_1, \ldots$.

There exists an algorithm which for a given $i \in \omega$ constructs the program P_i, and for a given P_i from this set finds the number i of this program in the considered sequence.

We will say that that we have an *enumeration* of all programs.

Enumeration of Programs

Let $i \in \omega$ and $k \in \omega \setminus \{0\}$. We denote by $\varphi_i^{(k)}$ the function of k variables that P_i computes.

Let P_i really implement the function $f(x_{i_1}, \ldots, x_{i_t})$. Then

$$\varphi_i^{(k)}(x_1, \ldots, x_k) = \begin{cases} f(x_1, \ldots, x_t), & \text{if } \ k \geq t, \\ f(x_1, \ldots, x_{k-1}, x_k, \ldots, x_k), & \text{if } \ k < t. \end{cases}$$

Enumeration of Programs

We denote by $t_i^{(k)}(x_1, \ldots, x_k)$ the time of work of the program P_i while computing the function $\varphi_i^{(k)}(x_1, \ldots, x_k)$.

Instead of $\varphi_i^{(1)}(x)$ we will write $\varphi_i(x)$.

Instead of $t_i^{(1)}(x)$ we will write $t_i(x)$.

Enumeration of Programs

We know that the set of partial recursive functions coincides with

$$\{\varphi_i^{(k)} : i \in \omega, k \in \omega \setminus \{0\}\}.$$

Here and later we use Church's thesis.

Existence of Complex Problems

First, we prove the existence of arbitrarily complex computable functions.

Theorem 10.1. *For any recursive function* $T(x)$*, there exists a recursive function* $f(x)$ *with values from* $\{0,1\}$ *such that, for any program* P_i *which computes* f *(for which* $\varphi_i(x) = f(x)$ *for any* $x \in \omega$*), there is an infinite number of values* $x \in \omega$ *such that* $t_i(x) > T(x)$*.*

Existence of Complex Problems

Proof. Let $g : \omega \to \omega$ be a recursive function which takes each value from ω infinite number of times.

For example, $g(x) = x - (\lfloor \sqrt{x} \rfloor)^2$.

We define the function f in the following way:

$$f(x) = \begin{cases} 1, & \text{if } t_{g(x)}(x) \leq T(x) \text{ and } \varphi_{g(x)}(x) = 0; \\ 0, & \text{otherwise.} \end{cases}$$

By Church's thesis, the function $f(x)$ is recursive.

Existence of Complex Problems

Let a program P_i compute the function $f(x)$, i.e., $f(x) = \varphi_i(x)$ for any $x \in \omega$.

Let j be an arbitrary number such that $g(j) = i$ (there is infinite number of such j). Let us assume that $t_i(j) \leq T(j)$.

Then, by definition of $f(x)$, we obtain the following: if $\varphi_i(j) = 0$ then $f(j) = 1$, and if $\varphi_i(j) \neq 0$ then $f(j) = 0$.

In any case, $f(j) \neq \varphi_i(j)$ which contradicts the assumption. Therefore $t_i(j) > T(j)$. $\square$

Existence of Complex Problems

Note that the function $T(x)$ can grow very quickly. For example, the function $h(n) = 2^{\left.2^{\cdots^{2}}\right\}n}$ is a recursive function.

Partial vs Total Functions

In this section, we will show that there exists a partial recursive function which can not be extended up to a total recursive function.

Partial vs Total Functions

Theorem 10.2 (on universal function). *There exists $z \in \omega$ such that for any $x, y \in \omega$*

$$\varphi_z^{(2)}(x, y) = \begin{cases} \varphi_x(y), & \text{if } \varphi_x(y) \text{ is defined,} \\ \text{undefined,} & \text{if } \varphi_x(y) \text{ is undefined.} \end{cases}$$

Partial vs Total Functions

Proof. For a given tuple $(x, y) \in \omega^2$, we find the program P_x and compute the value $\varphi_x(y)$.

So we have an algorithm for computation of the function

$$\psi(x, y) = \begin{cases} \varphi_x(y), & \text{if } \varphi_x(y) \text{ is defined}, \\ \text{undefined}, & \text{if } \varphi_x(y) \text{ is undefined}. \end{cases}$$

Partial vs Total Functions

By Church's thesis $\psi = \varphi_z^{(2)}$ for some $z \in \omega$. $\square$

This theorem can be generalized to the case of arbitrary number of variables.

Partial vs Total Functions

Theorem 10.3 (on impossibility of extension of a partial recursive function). *There exists a partial recursive function ψ such that any total recursive function f is not an extension of ψ.*

Partial vs Total Functions

Proof. Let $\varphi_z^{(2)}$ be an universal function, and let $\psi(x) = \varphi_z^{(2)}(x, x) + 1$, i.e.,

$$\psi(x) = \begin{cases} \varphi_x(x) + 1, & \text{if } \varphi_x(x) \text{ is defined,} \\ \text{undefined,} & \text{if } \varphi_x(x) \text{ is undefined.} \end{cases}$$

Let f be an arbitrary total recursive function and $f = \varphi_y$. Since f is total, the value $\varphi_y(y) = f(y)$ is defined. Hence, $\psi(y)$ is defined and $\psi(y) = \varphi_y(y) + 1 = f(y) + 1$. Therefore, f is not an extension of ψ. $\square$

Undecidable Problems

A problem is called *undecidable* if there is no algorithm which solves this problem.

Theorem 10.4 (on undecidability of halting problem). *The function*

$$g(x,y) = \begin{cases} 1, & \text{if } \varphi_x(y) \text{ is defined}, \\ 0, & \text{if } \varphi_x(y) \text{ is undefined}, \end{cases}$$

is not a total recursive function.

Undecidable Problems

Proof. Let us assume the contrary: g is a total recursive function.

We now define a function ψ as follows:

$$\psi(x) = \begin{cases} 1, & \text{if } g(x,x) = 0, \\ \text{undefined}, & \text{if } g(x,x) = 1. \end{cases}$$

By Church's thesis, ψ is a partial recursive function, and there exists $a \in \omega$ such that $\psi = \varphi_a$.

Undecidable Problems

Let us assume that the value $\varphi_a(a)$ is defined. Then $g(a, a) = 1$, $\psi(a)$ is undefined, but $\psi(a) = \varphi_a(a)$, contradiction!

Let us assume that the value $\varphi_a(a)$ is undefined. Then $g(a, a) = 0$, $\psi(a) = 1$, but $\psi(a) = \varphi_a(a)$, contradiction! $\square$

Undecidable Problems

We will generalize essentially this (the first) result about undecidability, but at the beginning we prove fixed-point theorem (Kleene, 1938).

Let f be a total recursive function, and $n \in \omega$. We will say that n is a *fixed-point value* for f if $\varphi_n = \varphi_{f(n)}$, where φ_i is the function implemented by the program P_i.

Theorem 10.5 (fixed-point theorem). *Let f be a total recursive function. Then there exists $n \in \omega$ such that*

$$\varphi_n = \varphi_{f(n)}.$$

Undecidable Problems

Proof. Let $u \in \omega$. We now describe a partial recursive function ψ_u by the following instructions. Let $x \in \omega$.

a). Apply the program P_u to u; if the result is undefined then the value $\psi_u(x)$ is undefined. Let the result of P_u work on u is w.

b). Apply P_w to x. If the result is undefined (P_w does not finish its work) then the value $\psi_u(x)$ is undefined. If P_w finish its work then the result of this work is $\psi_u(x)$.

Undecidable Problems

In the other words,

$$\psi_u(x) = \begin{cases} \varphi_{\varphi_u(u)}(x), & \text{if } \varphi_u(u) \text{ is defined}, \\ \text{undefined}, & \text{if } \varphi_u(u) \text{ is undefined}. \end{cases}$$

Undecidable Problems

So we describe an algorithm which for a given u constructs a program $P_{g(u)}$ which computes $\psi_u(x)$.

Using Church's thesis we obtain that g is a total recursive function. Thus

$$\varphi_{g(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x), & \text{if } \varphi_u(u) \text{ is defined,} \\ \text{undefined,} & \text{if } \varphi_u(u) \text{ is undefined.} \end{cases}$$

Undecidable Problems

Let f be an arbitrary total recursive function. Then $f(g(x))$ is a total recursive function.

Let $v \in \omega$ and $\varphi_v = f(g)$. Since $\varphi_v = f(g)$ is a total recursive function, the value $\varphi_v(v)$ is defined. From here it follows that

$$\varphi_{g(v)}(x) = \varphi_{\varphi_v(v)}(x) = \varphi_{f(g(v))}(x).$$

Thus, $n = g(v)$ is a fixed-point value for f. $\square$

Undecidable Problems

We denote by $PRF(1)$ the set of all partial recursive functions depending on one variable.

Let $F \subseteq PRF(1)$. We denote $N_F = \{n : \varphi_n \in F\}$. Here N_F is the set of numbers of functions φ_n belonging to F.

Let us consider the function

$$1 \dot{-} x = \begin{cases} 0, & \text{if } \ x \geq 1, \\ 1, & \text{if } \ x = 0. \end{cases}$$

This is a total recursive function. It can be obtained by the following scheme of primitive recursion:

$$\begin{cases} f(0) = s(0), \\ f(y+1) = 0. \end{cases}$$

Undecidable Problems

Theorem 10.6 (Rice, 1953). *Let $F \subseteq PRF(1)$, $F \neq \emptyset$ and $F \neq PRF(1)$. Then the set N_F is not a recursive set.*

Undecidable Problems

Proof. Let us assume the contrary: N_F is a recursive set. Then the function

$$f_{N_F}(x) = \begin{cases} 1, & \text{if } \ x \in N_F, \\ 0, & \text{if } \ x \notin N_F, \end{cases}$$

is a total recursive function.

Let us consider the function

$$f_{\bar{N}_F}(x) = \begin{cases} 1, & \text{if } \ x \in \bar{N}_F = \omega \setminus N_F, \\ 0, & \qquad \text{if } \ x \notin \bar{N}_F. \end{cases}$$

It is clear that $f_{\bar{N}_F} = 1 \dot{-} f_{N_F}$. Therefore $f_{\bar{N}_F}(x)$ is a total recursive function too.

Undecidable Problems

By assumption, $N_F \neq \emptyset$ and $\bar{N}_F \neq \emptyset$. Let $a \in N_F$ and $b \in \bar{N}_F$.

We denote $g(x) = af_{\bar{N}_F}(x) + bf_{N_F}(x)$. We have that

$$g(x) = \begin{cases} a, & \text{if} \quad x \in \bar{N}_F, \\ b, & \text{if} \quad x \in N_F, \end{cases}$$

is a total recursive function.

By fixed-point theorem, there exists $n \in \omega$ such that $\varphi_{g(n)} = \varphi_n$.

Undecidable Problems

It is clear that $n \in N_F$ or $n \in \bar{N}_F$.

1). Let $n \in N_F$. Then $\varphi_n \in F$ and $\varphi_{g(n)} \in F$. Since $n \in N_F$, $g(n) = b$, $b \in \bar{N}_F$ and $\varphi_{g(n)} \notin F$.

2). Let $n \notin N_F$. Then $\varphi_n \notin F$ and $\varphi_{g(n)} \notin F$. Since $n \notin N_F$, $g(n) = a$, $a \in N_F$ and $\varphi_{g(n)} \in F$.

So we obtain a contradiction. Thus, N_F is not a recursive set. $\square$

Semantic Properties of Programs

Semantic property of a program is a property connected with the function realized by the program. From Theorem of Rice it follows that each nontrivial semantic property of program is undecidable.

Nontrivial means that there exists a program which has this property, and there exists a program which has now this property.

Undecidable means that there is no algorithm that for a given program recognizes if the program has this property.

Semantic Properties of Programs

Let us consider some examples of nontrivial semantic properties.
To this end we should describe a nonempty set $F \subset PRF(1)$.

1. F_1 is the set of total recursive functions. The corresponding set of programs is the following: each program from this set on any $n \in \omega$ finishes its work and outputs a result.

2. F_2 is the set of partial recursive functions each of which is undefined only on a finite number of inputs from ω.

Semantic Properties of Programs

3. F_3 is the set of partial recursive functions each of which is defined on each $n \leq 2^{100}$.

4. F_4 is the set of partial recursive functions which are defined on 0.

5. F_5 is the set of constant functions.

Semantic Properties of Programs

6. F_6 is the set of partial recursive functions with an infinite set of values.

7. $F_7 = \{f\}$ where f is a given partial recursive function, for example, $f(x) = s(x) = x + 1$.

8. F_8 is the set of partial recursive functions which are equal to 1 on 1. Let us consider the last example in details. Let we test a program P_i on 1. What will be if our program works more than one month or one year on input 1? We do not know exactly if the value $\varphi_i(1)$ is defined.

Syntactical Properties of Programs

Any program is a finite word in some alphabet. Computable (decidable) properties of such words are *syntactical* properties of programs. Let us consider some examples:

1. The length of a program is at most 10^6.
2. The program has no loops.
3. The program does not use the operation of multiplication.

Discussion

Theorem of Rice does not mean that we can not do anything with semantic properties of programs.

It means only that for each nontrivial property there is no universal tool which is applicable to each program.