

Coding Programs by Numbers

Gödel numbers are usually very large, even for small programs.

Let us look at the following example:

[A] $X \leftarrow X+1$
 $\text{IF } X \neq 0 \text{ GOTO A}$

$\#(l_1) = \langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 21$
 $\#(l_2) = \langle 0, \langle 3, 1 \rangle \rangle = \langle 0, 23 \rangle = 46$

So the number of our small program is $2^{21} \cdot 3^{46} - 1$.

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 1

Coding Programs by Numbers

Note that the number of the unlabeled instruction $Y \leftarrow Y$ is $\langle 0, \langle 0, 0 \rangle \rangle = \langle 0, 0 \rangle = 0$.

Thus, the number of a program will be unchanged if an unlabeled instruction $Y \leftarrow Y$ is appended to it.

Although this ambiguity is harmless, we avoid it by adding a sentence to our definition of programs of \mathcal{L} :

The final instruction in a program is not permitted to be the unlabeled statement $Y \leftarrow Y$.

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 2

Coding Programs by Numbers

Then each number determines a **unique** program. As an example, let us determine the program whose number is 199:

$199 + 1 = 200 = 2^3 \cdot 3^0 \cdot 5^2 = [3, 0, 2]$.

So if $\#(\varphi) = 199$, φ consists of 3 instructions, the second of which is the unlabeled statement $Y \leftarrow Y$.

$3 = \langle 2, 0 \rangle = \langle 2, \langle 0, 0 \rangle \rangle$
 $2 = \langle 0, 1 \rangle = \langle 0, \langle 1, 0 \rangle \rangle$

Thus, the program is: [B] $Y \leftarrow Y$
 $Y \leftarrow Y$
 $Y \leftarrow Y+1$

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 3

The Halting Problem

Let us define the predicate **HALT(x, y)**.

For a given number y, let φ be the program such that $\#(\varphi) = y$.

Then HALT(x, y) is true if $\psi_{\varphi}^{(1)}(x)$ is defined and false if $\psi_{\varphi}^{(1)}(x)$ is undefined.

In other words:
 $\text{HALT}(x, y) \Leftrightarrow$ program number y eventually halts on input x.

Here comes a surprise:
Theorem 2.1: HALT(x, y) is not a computable predicate.

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 4

The Halting Problem

Proof (by contradiction):

Assume that HALT(x, y) were computable.

Then we could write the following program φ :

[A] $\text{IF HALT}(X, X) \text{ GOTO A}$

This program φ would compute the following function:

$\psi_{\varphi}^{(1)}(x) = \uparrow$ if $\text{HALT}(x, x)$
 $= 0$ if $\sim\text{HALT}(x, x)$

Now let $\#(\varphi) = y_0$. Then by the definition of HALT we get:

$\text{HALT}(x, y_0) \Leftrightarrow \sim\text{HALT}(x, x)$

For input $x = y_0$ we then have:

$\text{HALT}(y_0, y_0) \Leftrightarrow \sim\text{HALT}(y_0, y_0)$ **Contradiction!**

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 5

The Halting Problem

So finally we have an example of a predicate that is **not computable** by any program in the language \mathcal{L} .

We would even like to conclude the following:

There is **no algorithm** that, given a program of \mathcal{L} and an input to that program, can determine whether or not the given program will eventually halt on the given input.

This is called the **unsolvability of the halting problem**.

October 13, 2009 Theory of Computation
 Lecture 11: A Universal Program III 6

The Halting Problem

If there were such an algorithm, we could use it to determine the truth value of $\text{HALT}(x, y)$ for given x and y :

- We would first obtain the program Q so that $\#(Q) = y$.
- Then we would check whether Q eventually halts on input x .

However, we have reason to believe that any algorithm for computing on numbers can be carried out by a program of \mathcal{L} (**Church's Thesis**).

So this would contradict the fact that $\text{HALT}(x, y)$ is not computable.

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

7

The Halting Problem

It may surprise you that there is no algorithm for solving the halting problem.

Is it not possible for a computer scientist to analyze a given program and find out whether it will terminate for a particular input or not (even if this analysis takes a very long time)?

No, actually we can devise a very simple program in \mathcal{L} for which to date nobody is able to tell whether it will ever terminate.

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

8

The Halting Problem

This program is based on **Goldbach's conjecture**, which assumes that every even number ≥ 4 is the sum of two prime numbers.

For example, $4 = 2 + 2$, $6 = 3 + 3$, $48 = 41 + 7$.

It would be easy to write a program in \mathcal{L} that searches for a counterexample to this conjecture.

This program would check the following predicate for increasing values n :

$$\sim(\exists x)_{\leq n}(\exists y)_{\leq n}[\text{Prime}(x) \ \& \ \text{Prime}(y) \ \& \ x + y = n]$$

Nobody knows whether this program will ever halt.

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

9

Universality

For each $n > 0$, let us define:

$$\Phi^{(n)}(x_1, \dots, x_n, y) = \Psi_{\varphi^{(n)}}(x_1, \dots, x_n), \quad \text{where } \#(\varphi) = y.$$

Theorem 3.1 (Universality Theorem):

For each $n > 0$, the function $\Phi^{(n)}(x_1, \dots, x_n, y)$ is partially computable.

This is **one of the most important theorems** in computability theory.

We will prove it by providing instructions for writing a program \mathcal{U}_n that computes $\Phi^{(n)}$ for each $n > 0$.

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

10

Universality

In other words, for each $n > 0$ we want to have:

$$\Psi_{\mathcal{U}_n^{(n+1)}}(x_1, \dots, x_n, x_{n+1}) = \Phi^{(n)}(x_1, \dots, x_n, x_{n+1}).$$

These programs \mathcal{U}_n are called universal.

For example, \mathcal{U}_1 can be used to compute any partially computable function of one variable.

If there is a program φ that computes $f(x)$, and $\#(\varphi) = y$, then $f(x) = \Phi^{(1)}(x, y) = \Psi_{\mathcal{U}_1^{(2)}}(x, y)$.

It is useful to think of the programs \mathcal{U}_n in terms of **interpreters** of programs in \mathcal{L} .

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

11

Universality

The universal programs must

- decode the number of the program given to them,
- keep track of the current snapshot during program execution, and
- generate the next snapshot based on the current one and the current instruction.

When we write such programs, we will freely use macros referring to functions that we already know to be primitive recursive.

We will also freely use label and variable names beyond those specified for the language \mathcal{L} .

October 13, 2009

Theory of Computation
Lecture 11: A Universal Program III

12

Universality

In describing the state of a computation, we assume all variables to have the value 0 if not assigned a different value.

Then we can code the state of the computation by the Gödel number $[a_1, \dots, a_m]$, where a_i is the value of the i -th variable in our ordered list.

Obviously, m is chosen so that all a_i for $i > m$ are 0.

For example, the state $Y = 1, X = 2, Z_2 = 1$ is coded by the following number:

$$[1, 2, 0, 0, 1] = 2 \cdot 3^2 \cdot 11 = 198.$$

October 13, 2009

Theory of Computation Lecture 11: A Universal Program III

13

Universality

In order to store the current snapshot, we need to keep track of two numbers:

- K is the **number of the instruction** to be executed next, and
- S is the **current state** coded as a Gödel number (see previous slide).

Now we are ready to write the program \mathcal{U}_n for computing $Y = \Phi^{(n)}(X_1, \dots, X_n, X_{n+1})$.

We will explain \mathcal{U}_n piece by piece and finally put the pieces together.

October 13, 2009

Theory of Computation Lecture 11: A Universal Program III

14

Universality

$$Z \leftarrow X_{n+1} + 1$$

$$S \leftarrow \prod_{i=1}^n (p_{2i})^{X_i}$$

$$K \leftarrow 1$$

If $X_{n+1} = \#(\mathcal{P})$, where \mathcal{P} consists of the instructions l_1, \dots, l_m , then $Z = [\#(l_1), \dots, \#(l_m)]$.

S is initialized as $[0, X_1, 0, X_2, \dots, 0, X_n]$, which puts the input values into the first n input variables and 0s into the other variables.

K is given the value 1 so that the computation will begin with the first instruction.

October 13, 2009

Theory of Computation Lecture 11: A Universal Program III

15

Universality

Then we append the following instruction:

[C] IF $K = \text{Lt}(Z) + 1 \vee K = 0$ GOTO F

So if the computation has ended, GOTO F, where the proper value will be output.

Otherwise, the current instruction is decoded and executed:

$$U \leftarrow r((Z)_K)$$

$$P \leftarrow p_{r(U)+1}$$

Remember that $(Z)_K = \langle a, \langle b, c \rangle \rangle$ is the number of the K -th instruction.

So $U = \langle b, c \rangle$ is the code of the statement to be executed.

The variable mentioned in this statement is the $(c + 1)$ -th, i.e., the $(r(U) + 1)$ -th.

October 13, 2009

Theory of Computation Lecture 11: A Universal Program III

16