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 \([a_1, \ldots, a_m]\), 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:

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

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 \(U_n\) for computing \(Y = \Phi(n)(X_1, \ldots, X_n, X_{n+1})\).

We will explain \(U_n\) piece by piece and finally put the pieces together.

Universality

\[
Z \leftarrow X_{n+1} + 1
\]
\[
S \leftarrow \prod_{i=1}^{n} (p_{a_i})^{X_i}
\]
\[
K \leftarrow 1
\]

If \(X_{n+1} = \#(P)\), where \(P\) consists of the instructions \(I_1, \ldots, I_m\), then \(Z = \#(I_1), \ldots, \#(I_m)\).

\(S\) is initialized as \([0, X_1, 0, X_2, \ldots, 0, X_n]\), 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.

Universality

Then we append the following instruction:

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

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

Otherwise, the current instruction is decoded and executed:

\[
U \leftarrow r((Z)K)
\]
\[
P \leftarrow p_{a_{(Z)K}}
\]

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

So \(U = (b, c)\) 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.

Universality

Then \(l(U)\) contains the type of the instruction to be executed.

\[
\text{IF } l(U) = 0 \text{ GOTO N}
\]
\[
\text{IF } l(U) = 1 \text{ GOTO A}
\]
\[
\text{IF } \sim (P | S) \text{ GOTO N}
\]
\[
\text{IF } l(U) = 2 \text{ GOTO M}
\]

If neither the instruction is \(V \leftarrow V\), or the instruction is \(V \leftarrow V-1\) and \(V = 0\) (as indicated by the absence of \(P\) in \(S\)), or the instruction is \(\text{IF } V \neq 0 \text{ GOTO L}\) and \(V = 0\), then nothing is done to \(S\) ("nothing" – GOTO N).

If the instruction is \(V \leftarrow V+1\), then the exponent of \(P\) in \(S\) needs to be incremented ("add" – GOTO A).

If the instruction is \(V \leftarrow V-1\) with \(V > 0\), then the exponent of \(P\) in \(S\) needs to be decremented ("minus" – GOTO M).

Universality

If none of the four previous predicates were true, a GOTO command has to be executed:

\[
K \leftarrow \min \{l(U)|l(U) + 2 = l(V)\}
\]

GOTO C

So if the label \(l(U) + 2\) exists in the program, the number \(K\) of the next instruction to be executed will be set to the first instruction with that label.

Otherwise, \(K\) will be set to 0.

As you remember, if \(K = 0\) or \(K = \text{Lt}(Z) + 1\), then the computation stops.

In any case, our interpreter program executes a GOTO C to execute the next instruction.
Universality

The program continues as follows:

[M] \( S \leftarrow \lfloor S/P \rfloor \)
GOTO N

[A] \( S \leftarrow S \cdot P \)

[N] \( K \leftarrow K + 1 \)
GOTO C

The value of the variable in the current instruction is decremented or incremented by 1 by dividing or multiplying \( S \) by \( P \), respectively.
Then \( K \) is incremented and the next instruction executed.

Universality

The program concludes with the following line:

[F] \( Y \leftarrow (S)_1 \)

This way, after termination of the interpreted program, its output value becomes the output value of the interpreter.

On the next slide, we will list the entire interpreter program.

Universality

For each \( n > 0 \), the sequence

\[ \Phi^{(n)}(x_1, \ldots, x_n, y, 1), \ldots \]

enumerates all partially computable functions of \( n \) variables. We can also write:

\[ \Phi^{(n)}(x_1, \ldots, x_n) = \Phi^{(n)}(x_1, \ldots, x_n, y). \]

We can omit the superscript \( (n) \) when \( n = 1 \):

\[ \Phi(x) = \Phi(x, y) = \Phi^{(1)}(x, y). \]

Universality

Consider the following predicates:

\[ \text{STP}^{(n)}(x_1, \ldots, x_n, y, t) \]

\( \Leftrightarrow \) Program number \( y \) halts after \( t \) or fewer steps on inputs \( x_1, \ldots, x_n \)

\( \Leftrightarrow \) There is a computation of program \( y \) of length \( \leq t + 1 \), beginning with inputs \( x_1, \ldots, x_n \)

These predicates are computable, which we can easily prove:

We can simply add a counter to our universal programs to determine when we have simulated \( t \) steps.

Universality

We can prove an even stronger theorem:

**Theorem 3.2 (Step-Counter Theorem):**

For each \( n > 0 \), the predicate \( \text{STP}^{(n)}(x_1, \ldots, x_n, y, t) \) is primitive recursive.

**Proof:**

We will provide numerical descriptions of the notions of snapshot and successor snapshot.
This will show that the necessary functions are primitive recursive.
See pages 74 and 75 in the textbook for the proof.
Universality

Now that we know the Step-Counter Theorem, we are ready for yet another theorem.
Proving this theorem will be similar to the last one.

**Theorem 3.3 (Normal Form Theorem):**
Let \( f(x_1, \ldots, x_n) \) be a partially computable function.
Then there is a primitive recursive predicate \( R(x_1, \ldots, x_n, y) \) such that
\[
f(x_1, \ldots, x_n) = l(\min_z R(x_1, \ldots, x_n, z))
\]

**Proof:**
Let \( y_0 \) be the number of a program that computes \( f(x_1, \ldots, x_n) \).
Then consider the equation
\[
f(x_1, \ldots, x_n) = l(\min_z R(x_1, \ldots, x_n, z))
\]
where \( R(x_1, \ldots, x_n, z) \) is the predicate
\[
\text{STP}(n)(x_1, \ldots, x_n, y_0, \text{r}(z)) \land (\text{r}(\text{SNAP}(n)(x_1, \ldots, x_n, y_0, \text{r}(z)))) = l(z)
\]
If the right side of the first equation is **defined**, then there exists a number \( z \) such that
\[
\text{STP}(n)(x_1, \ldots, x_n, y_0, \text{r}(z)) \land (\text{r}(\text{SNAP}(n)(x_1, \ldots, x_n, y_0, \text{r}(z)))) = l(z)
\]
For any such \( z \),
- the computation by the program with number \( y_0 \) has reached a terminal snapshot in \( r(z) \) or fewer steps,
- \( l(z) \) is the value of the output variable \( Y \), that is, \( l(z) = f(x_1, \ldots, x_n) \).

If the right side of the equation is **undefined**, it must be true that \( \text{STP}(n)(x_1, \ldots, x_n, y_0, t) \) is false for all values of \( t \), that is, \( f(x_1, \ldots, x_n) \) is undefined.