THEORY OF COMPUTATION
Coding Programs - 8

Prof. Dan A. Simovici

UMB

Each program $\mathcal{P}$ of $\mathcal{S}$ will receive a number $\#(\mathcal{P})$ such that the program can be retrieved from this number.

CODING VARIABLES and LABELS:

- variables will be arranged in the following order:

$$Y, X_1, Z_1, X_2, Z_2, X_3, Z_3, \ldots, X_i, Z_i, \ldots$$

- labels will be arranged as

$$A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, A_3, \ldots$$

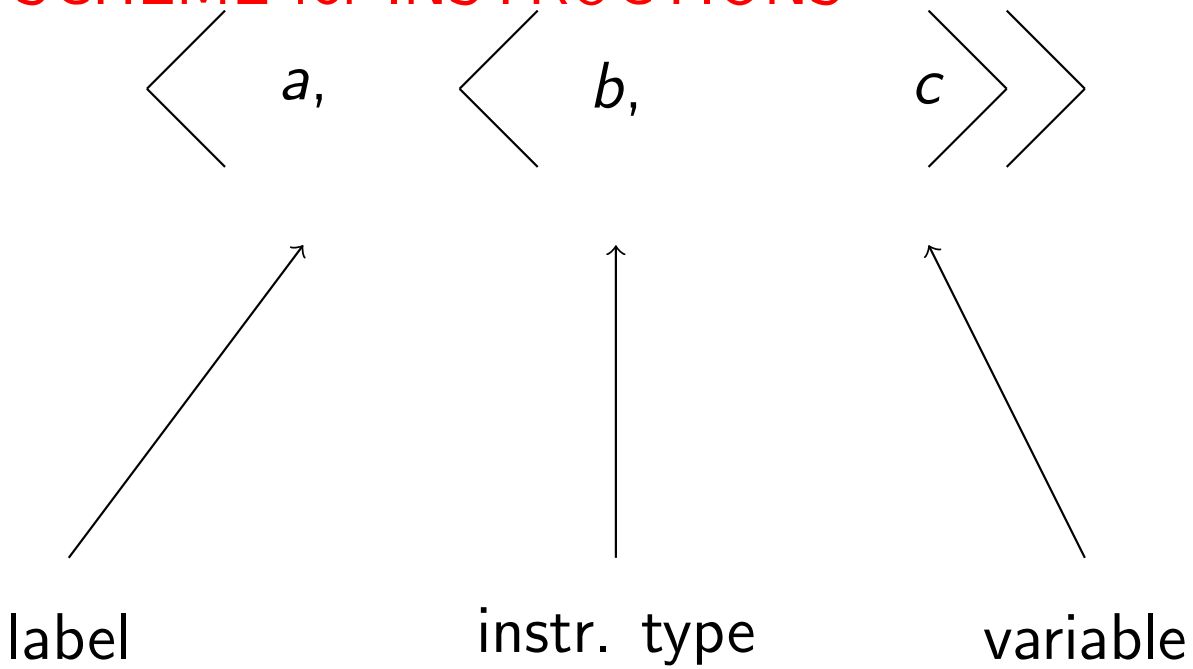
$\#(V)$ and $\#(L)$ are the positions of a given variable or a given label in the ordering.

Example

We have

$$\#(X_2) = 4, \#(Z_1) = \#(Z) = 3, \#(E_1) = \#(E) = 5, \#(B_2) = 7.$$

CODING SCHEME for INSTRUCTIONS



CODING INSTRUCTIONS:

Let I be an instruction, labeled or unlabeled of $\mathcal{S}$.
We write $\#(I) = \langle a, \langle b, c \rangle \rangle$, where

- if I is an unlabeled, then $a = 0$; if I la labeled L, then $a = \#(L)$;
- if the variable V is mentioned in I, then $c = \#(V) - 1$;
- if the statement is $V \leftarrow V$, or $V \leftarrow V + 1$, or $V \leftarrow V - 1$, then $b = 0$, or 1, or 2, respectively;
- if the statement is IF $V \neq 0$ GOTO L', then $b = \#(L') + 2$.

Example

The code of the unlabeled instruction

$$X \leftarrow X + 1$$

is

$$\langle 0, \langle 1, 1 \rangle \rangle = \langle 0, 5 \rangle = 10.$$

Example

The code of the labeled instruction

$$[A] \;\; X \leftarrow X + 1$$

is

$$\langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 2^1(2 \cdot 5 + 1) - 1 = 21.$$

For any given q there is a unique instruction I with $\#(I) = q$.

To decode an instruction:

- compute $\ell(q)$: if $\ell(q) = 0$, I is unlabeled; otherwise I has the $\ell(q)^{\text{th}}$ label on the list.
- to find the variable mentioned in I, compute $i = r(r(q)) + 1$ and locate the i^{th} variable on the list;
- to find the type of the instruction compute $\ell(r(q))$.

Instruction Type:

$\ell(r(q))$	Instruction
0	$V \leftarrow V$
1	$V \leftarrow V + 1$
2	$V \leftarrow V - 1$
$j = \ell(r(q)) - 2$	IF $V \neq 0$ GOTO L where L is the jth label

CODING PROGRAMS:

If a program $\mathcal{P}$ consists of instructions

$$I_1, I_2, \ldots, I_n,$$

then its Gödel number is

$$\#(\mathcal{P}) = [\#(I_1), \#(I_2), \ldots, \#(I_n)] - 1.$$

Gödel numbers can be very large! Thus, the code of simple programs can be quite enormous.

Example

For the program $\mathcal{P}$:

$$\begin{array}{ll} [A] & X \leftarrow X + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \end{array}$$

that consists of the instructions I_1 and I_2 we have $\#(I_1) = 21$ (as we saw) and $\#(I_2) = \langle 0, \langle 3, 1 \rangle \rangle = 46$. Therefore, the Gödel number of the program is $\#(\mathcal{P}) = 2^{21} \cdot 3^{46} - 1$.

Example

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 Gödel number of a program remains the same if an unlabeled statement $Y \leftarrow Y$ is added onto its end. This is a harmless ambiguity because $Y \leftarrow Y$ added at the end of a program does nothing. Also, this is harmless we add the stipulation that the final instruction of a program cannot be $Y \leftarrow Y$. With this stipulation each Gödel number determines a unique program.

Example

Let us determine the program whose number is 199. We have

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

Thus, the program consists of three instruction having the codes 3,0 and 2, respectively.
We have $3 = \langle 2, 0\rangle = \langle 2, \langle 0, 0\rangle\rangle$, and $2 = \langle 0, 1\rangle = \langle 0, \langle 1, 0\rangle\rangle$. This implies that the program is

$$\begin{array}{ll} [B] & Y \leftarrow Y \\ & Y \leftarrow Y \\ & Y \leftarrow Y + 1 \end{array}$$

The program computes the constant function $y = 1$.

Example

The empty program has the number $1 - 1 = 0$.

Definition

Let HALT be the predicate:

$$\text{HALT}(x, y) = \begin{cases} 1 & \text{if the program } \mathcal{P} \text{ with } \#(\mathcal{P}) = y \text{ halts on input } x, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\text{HALT}(x, y) = 1$ if $\psi_{\mathcal{P}}^{(1)}(x)$ is defined and $\text{HALT}(x, y) = 0$ if $\psi_{\mathcal{P}}^{(1)}(x)$ is undefined.

Theorem

$HALT(x, y)$ is not a computable predicate.

Proof.

Suppose that HALT were computable. Then, we could construct the program $\mathcal{P}$:

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

We have

$$\psi_{\mathcal{P}}(x) = \begin{cases} \text{undefined} & \text{if } HALT(X, X), \\ 0 & \text{if } \sim HALT(X, X). \end{cases}$$

Let $\#(\mathcal{P}) = y_0$. Using the definition of HALT we have

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

for all x. If we take $x = y_0$, then $\text{HALT}(y_0, y_0) \Leftrightarrow \sim \text{HALT}(y_0, y_0)$, which is a contradiction! □

Conclusions:

- The previous theorem provides an example of a function that is not computable by any program in $\mathcal{S}$.
- There is no algorithm that given a program in $\mathcal{S}$ and an input to that program can determine whether or not the given program will eventually halt on the given input. This is unsolvability of the halting problem.

The Church's Thesis: any algorithm for computing on numbers can be carried out by a program in $\mathcal{S}$.

- Church thesis cannot be proven as a mathematical theorem because the notion of algorithm has no general definition separated from a programming language.
- Thus, Church's thesis allows us to assert the non-existence of algorithms whenever we have shown that some problem cannot be solved by a program in $\mathcal{S}$.

It is easy to construct short programs (in $\mathcal{S}$) such that nobody is in a position to tell if they will eventually hold.

Example

Goldbach's conjecture: is one of the oldest and best-known unsolved problems in number theory and all of mathematics. It states that every even integer greater than 2 is the sum of two primes:

$$4 = 2 + 2, 6 = 3 + 3, 8 = 3 + 5, \ldots$$

Example

To check if an even number n is a counterexample requires checking a primitive recursive predicate:

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

Example cont'd

Example

This predicate allows us to write a program $\mathcal{P}$ in $\mathcal{S}$ that will search for a counterexample to Goldbach conjecture, that is, an even number $n \geqslant 4$ that is not the sum of two primes:

$$\begin{array}{ll} & Z \leftarrow 4 \\ [A] & \text{IF } G(Z) = 0 \text{ GOTO } E \\ & Z \leftarrow Z + 1 \\ & \text{GOTO } A \end{array}$$

The statement that $\mathcal{P}$ never halts is equivalent to the Goldbach conjecture. Since this conjecture is still open after 250 years, nobody knows that the program $\mathcal{P}$ will eventually halt!

Christian Goldbach was born in the Duchy of Prussia's capital Königsberg, part of Brandenburg-Prussia in 1690. Goldbach was the son of a pastor, studied at the Royal Albertus University, and after finishing his studies he went on long educational voyages from 1710 to 1724 through Europe.
He worked at the St. Petersburg Academy of Sciences in 1725, as a professor of mathematics and historian of the academy. In 1728, when Peter II became Tsar of Russia, Goldbach became his tutor. He corresponded with Euler and the famous conjecture was stated in one of his letters.
He died on November 20, 1764 at age of 74, in Moscow.