The Programming Language $L$

We will explore computability theory using an extremely simple programming language called $L$.
Let us take a look at this language. $L$ uses variables holding numbers. Throughout the course, ‘number’ will refer to a nonnegative integer.
The variables named $X_1, X_2, X_3, \ldots$ are the input variables of $L$, $Y$ is the output variable of $L$, and $Z_1, Z_2, Z_3, \ldots$ are the local variables of $L$.

The Programming Language $L$

We do not have to write the subscript 1, so instead of $X_1$ or $Z_1$ we can write $X$ or $Z$.

$L$ also includes labels. These labels are named $A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, \ldots$

Again, the subscript 1 can be omitted.

The Programming Language $L$

A program of $L$ consists of a list (a finite sequence) of instructions.

What do the instructions look like?

There are only three types of instructions in $L$:

- Increment
- Decrement
- Conditional branch

The Programming Language $L$

In the following list of instructions, the letter $V$ stands for any variable in the program, and $L$ stands for a label:

$V \gets V+1$ increase by 1 the value of the variable $V$

$V \gets V-1$ If the value of $V$ is 0, leave it unchanged; otherwise decrease by 1 the value of the variable $V$.

If $V \neq 0$ GOTO $L$ If the value of $V$ is nonzero, perform the instruction with label $L$ next; otherwise proceed to the next instruction in the list.

The Programming Language $L$

Note that we have an unlimited supply of variables and labels.

Moreover, there is no upper limit on the value that a variable can contain.

Therefore, the language $L$ is not a practical language, but it is well-suited for the theoretical evaluation of algorithms.

The Programming Language $L$

These are the only three instructions in our language $L$.
You will be surprised how powerful this extremely simple programming language is.

We just need two more conventions:

- The output variable $Y$ and the local variables $Z_i$ initially have the value 0.
- A program halts when it attempts to move to a nonexistent instruction (beyond the end of the list) or branch to a nonexistent label.
Sample Programs

Consider the following program:

[A]  \( X \leftarrow X-1 \)
     \( Y \leftarrow Y+1 \)
     IF \( X \neq 0 \) GOTO A

What function does this program compute?
\( f(x) = 1, \text{ if } x = 0 \)
\( = x, \text{ otherwise.} \)

What would we have to change if we wanted to compute the function \( f(x) = x \)?

Sample Programs

The following program computes \( f(x) = x \):

[A]  IF \( X=0 \) GOTO B
     \( Z \leftarrow Z+1 \)
     IF \( Z=0 \) GOTO E
 [B]  \( X \leftarrow X-1 \)
     \( Y \leftarrow Y+1 \)
     IF \( X=0 \) GOTO B

Sample Programs

In the previous example, the lines
\( Z \leftarrow Z+1 \)
IF \( Z=0 \) GOTO L
were used to implement an instruction that we could call
GOTO L
First, we make sure that \( Z \) has a nonzero value, and then the conditional branch (actually, here it is an unconditional branch) is executed.

Sample Programs

From now on we will use the macro GOTO L in our programs.
We know that we could always replace GOTO L with its macro expansion to obtain a valid \( L \) program.
Now remember the program computing the function \( f(x) = x \).
Although it computes its output correctly, it deletes (sets to zero) the original input.
This is an undesirable behavior, so we should come up with an improved program.

Sample Programs

The previous program justifies the introduction of a macro
\( V \leftarrow V' \)
The execution of this macro will replace the contents of variable \( V \) by those of variable \( V' \) without changing the contents of \( V' \).
However, there is one problem:
The previous program could rely on the initial condition \( Y = 0 \), which is not guaranteed for any variable \( V \).
Sample Programs

To solve this problem, we introduce the macro
V$\leftarrow 0$
Its macro expansion is:
[L] \ V$\leftarrow \ V - 1$
\quad \text{IF } V \neq 0 \text{ GOTO L}

Of course, the label L has to be chosen to be different from any other label in the program.
Now we can write down the macro expansion of V$\leftarrow V'$:

Another example: \( f(x_1, x_2) = x_1 + x_2 \)

Y$\leftarrow X_1$
Z$\leftarrow X_2$
[B] \ IF Z \neq 0 \text{ GOTO A}
\quad \text{GOTO E}
[A] \ Z \leftarrow Z - 1$
\quad \ Y \leftarrow Y + 1$
\quad \text{GOTO B}

The Syntax of \( \mathcal{L} \)

We will now develop a precise mathematical description of the programming language \( \mathcal{L} \).
The symbols \( X_1, X_2, X_3, \ldots \)
are called **input variables,**
\( Z_1, Z_2, Z_3, \ldots \)
are called **local variables,**
and \( Y \) is called the **output variable** of \( \mathcal{L} \).

The symbols
A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, \ldots
are called **labels** of \( \mathcal{L} \).

For variables and labels, the subscript 1 can always be omitted.

A **statement** is one of the following:

\( V \leftarrow V + 1 \)
\( V \leftarrow V - 1 \)
\( V \leftarrow V \)
\( \text{IF } V \neq 0 \text{ GOTO L} \)

Here, \( V \) may be any variable and \( L \) may be any label.
Note that the statement \( V \leftarrow V \) leaves all values unchanged, so it is a "dummy" command.
We will later see why it is useful to have \( V \leftarrow V \) in our set of statements.
The Syntax of $L$

An instruction is either
• a statement (unlabeled instruction) or
• [L] followed by a statement (instruction labeled L)

A program is a list (i.e., a finite sequence) of instructions.
The length of this list is called the length of the program.
We also include the empty program – the program of length 0 – in the set of all programs.

The Syntax of $L$

While a program is being executed, its variables assume different numerical values.
This motivates the concept of the state of a program:
A state of a program $\varphi$ is a list of equations of the form $V = m$,
where $V$ is a variable and $m$ is a number,
including exactly one equation for each variable that occurs in $\varphi$ (and possibly equations for other variables).

The Syntax of $L$

Consider the following program $\varphi$:

[A] IF $X \neq 0$ GOTO B
    Z ← Z + 1
    IF $Z = 0$ GOTO E
[B] X ← X - 1
    Y ← Y + 1
    Z ← Z + 1
    IF $Z \neq 0$ GOTO A

Is the list $X = 4, Y = 3, Z = 3$ a state of $\varphi$?
Yes.
How about $X_1 = 4, X_2 = 5, Y = 4, Z = 4$?
Yes.
And $X = 3, Z = 3$?
No.

The Syntax of $L$

Let $\sigma$ be a state of $\varphi$ and let $V$ be a variable that occurs in $\sigma$.
The value of $V$ at $\sigma$ is then the unique number $q$ such that the equation $V = q$ is one of the equations in $\sigma$.
For example, the value of $X$ at the state $X = 4, Y = 3, Z = 3$ is 4.

The Syntax of $L$

Consider a machine that can execute programs of $L$.
During program execution, this machine needs to store the state of a program, i.e., keep track of the values of variables.
Is there anything else that needs to be stored?
Yes, we also need to store which instruction is to be executed next.

The Syntax of $L$

Therefore, we define a snapshot or instantaneous description of a program $\varphi$ of length $n$ to be a pair $(i, \sigma)$ where $1 \leq i \leq (n + 1)$, and $\sigma$ is a state of $\varphi$.
The number $i$ indicates the number of the instruction that is to be executed next.
$i = n + 1$ corresponds to a "stop" instruction.
A snapshot $(i, \sigma)$ of a program $\varphi$ of length $n$ is called terminal if $i = n + 1$.
If $s = (i, \sigma)$ is a snapshot of $\varphi$ and $V$ is a variable of $\varphi$,
then the value of $V$ at $s$ just means the value of $V$ at $\sigma$. 
The Syntax of $L$

If $(i, \sigma)$ is a nonterminal snapshot of $P$, we define the successor of $(i, \sigma)$ to be the snapshot $(j, \tau)$ defined as follows:

**Case 1:**
The i-th instruction of $P$ is $V \leftarrow V+1$ and $\sigma$ contains the equation $V = m$.
Then $j = i + 1$ and $\tau$ is obtained from $\sigma$ by replacing the equation $V = m$ with $V = m + 1$ (the value of $V$ at $\tau$ is $m + 1$).

**Case 2:**
The i-th instruction of $P$ is $V \leftarrow V-1$ and $\sigma$ contains the equation $V = m$.
Then $j = i + 1$ and $\tau$ is obtained from $\sigma$ by replacing the equation $V = m$ with $V = m - 1$ if $m \neq 0$.
If $m = 0$, then $\tau = \sigma$.

**Case 3:**
The i-th instruction of $P$ is $V \leftarrow V$.
Then $j = i + 1$ and $\tau = \sigma$.

**Case 4:**
The i-th instruction of $P$ is IF $V \neq 0$ GOTO L.
Then $\tau = \sigma$ and there are two subcases:

**Case 4a:**
$\sigma$ contains the equation $V = 0$.
Then $j = i + 1$.

**Case 4b:**
$\sigma$ contains the equation $V = m$ where $m \neq 0$.
Then, if there is an instruction of $P$ labeled L, $j$ is the least number such that the j-th instruction of $P$ is labeled L. Otherwise, $j = n + 1$. 

---

2/1/2019