ECE 664
Lecture 3

Avi KAK
Purdue Univ

A Minimalist Programming Language $\mathcal{S}$

- The instruction set of $\mathcal{S}$ consists of just the following three statements:

```
V ← V+1
V ← V-1
IF V≠0, GO TO L
```

V remains unchanged if V=0

As you'll see eventually, practically everything that modern computers do can be boiled down to computing using just these three instructions.

It's interesting to reflect on the fact that there are many "famous" people who are convinced that computer intelligence will some day exceed human intelligence. One would think there is infinitely more to human intelligence than the three instructions shown above.

- In order to demonstrate the power of $\mathcal{S}$, we will use the following conventions:

① We denote:
- Input variables by: $X, X_1, X_2 \ldots$
- Output variable by: Y
- Local variables by: $Z, Z_1, Z_2, \ldots$
- Labels by: $A, B, C, \ldots, A_1, B_1, C_1, \ldots$
- A labeled statement by: [B] $Z \leftarrow Z+1$

② We assume that all local variables and the output variable are set to 0 initially.

③ We require that only the set $N = \{0, 1, 2, \ldots\}$ of natural numbers be used to initialize the input variables and for calculations with $\mathcal{S}$.

④ We will use $\mathcal{P}$ to denote a program made up of the statements of $\mathcal{S}$.

Some Starter Programs in $\mathcal{S}$

- My goal in the rest of this Lecture is to demonstrate some simple programs in $\mathcal{S}$. The lectures that follow will go into the power of $\mathcal{S}$.
- Let's say we want to compute $y = x$. That is, we want the output variable Y to acquire the value of the input variable X. For our first attempt, we could try

```
[A] X ← X-1
    Y ← Y+1
    IF X≠0, GO TO A
```

This program actually computes $y = \begin{cases} 1 & x = 0 \\ x & \text{otherwise} \end{cases}$ which is not what we want.

- For our next attempt, we could try:

```
[A] IF X≠0 GO TO B
    Z ← Z+1
    IF Z≠0 GO TO E
[B] X ← X-1
    Y ← Y+1
    Z ← Z+1
    IF Z≠0 GO TO A
```

can be represented by the macro GOTO E

can be represented by the macro GOTO A

We could say that 'E' stands for "stop the program and read Y"

E stands for the Exit Label.

In general,

```
Z ← Z+1
IF Z≠0 GO TO L
```

is the macro expansion of GO TO L

- While the second program shown on the previous page does solve the $y = x$ problem, it does suffer from a "computational" shortcoming: It destroys the value stored in X. The next program (that uses the macro "GOTO L" for brevity) eliminates this shortcoming:

```
[A] IF X ≠ 0 GO TO B
    GOTO C          (macro)
[B] X ← X − 1
    Y ← Y + 1
    Z ← Z + 1
    GO TO A
[C] IF Z ≠ 0 GO TO D
    GOTO E
[D] Z ← Z − 1
    X ← X + 1
    GOTO C          (macro)
```

This program correctly computes $y \equiv f(x) = x$ and does so without destroying the input variable X.

In the rest of this lecture, we will use this program as a macro for $V \leftarrow V'$. Using this program as a macro will, in general, require that we first initialize V by $V \leftarrow 0$. Thinking of $V \leftarrow 0$, its macro expansion is:

```
[L] V ← V − 1
    IF V ≠ 0 GOTO L
```

Some Higher Level Functions Programmed in $\mathcal{S}$

- Using the two macros defined above, "$V \leftarrow V'$" and the unconditional "GOTO L", we can now more easily write some higher level programs in $\mathcal{S}$. Here is a program that computes the function $f(x_1, x_2) = x_1 + x_2$:

```
    Y ← X1
    Z ← X2
[B] IF Z ≠ 0 GOTO A
    GO TO E
[A] Z ← Z − 1
    Y ← Y + 1
    GO TO B
```

We use Z to preserve the value of X_2

← We will use this program as a macro for $V \leftarrow V_1 + V_2$

- Here is a program in $\mathcal{S}$ for computing the function $f(x_1, x_2) = x_1 \cdot x_2$:

```
    Z2 ← X2
[B] IF Z2 ≠ 0 GOTO A
    GOTO E
[A] Z2 ← Z2 − 1
    Z1 ← X1 + Y
    Y  ← Z1
    GO TO B
```

To preserve X_2

based on the idea that multiplication is nothing but repeated additions

- The program shown below computes the partial function $g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ \uparrow & \text{if } x_1 < x_2 \end{cases}$

means undefined

```
    Y ← X1
    Z ← X2
[C] IF Z ≠ 0 GOTO A
    GOTO E
[A] IF Y ≠ 0 GOTO B
    GOTO A
[B] Y ← Y − 1
    Z ← Z − 1
    GOTO C
```

- The important point here is: How does one capture the notion of "undefined" in a program in $\mathcal{S}$?
- In general, you cannot write a program to meet every possible contingency vis-a-vis the input variables. (In practice, programs throw exceptions when they encounter conditions for which program execution was not designed.)
- Note how this program enters an infinite loop when the data condition $X_1 < X_2$ is encountered.