بسم الله الرحمن الرحیم

Theory of Computer Science

Ali Shakiba

Vali-e-Asr University of Rafsanjan

ali.shakiba@vru.ac.ir

What we are going to discuss?

Theory of Computation

Complexity Theory

another topic …

- Mining Massive Datasets
- Computational Learning Theory
- Parameterized Algorithms

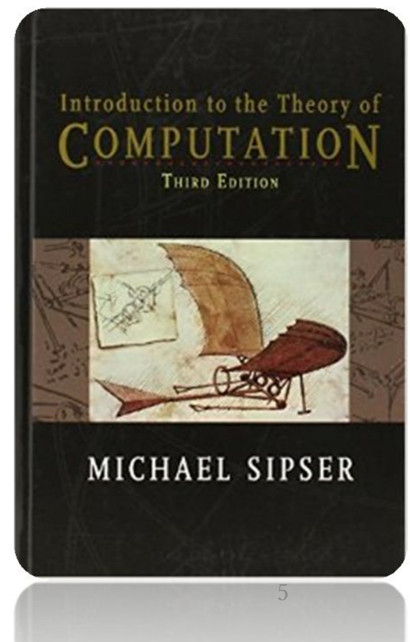
Theory of Computation

- Church-Turing thesis, Turing machine & its variations
- exploring the limits of algorithmic solvability
- reducibility as a key method to prove unsolvability
- recursive/partial recursive functions
- decidability in terms of recursion
- arriving at Turing machines
- decidability of logical theories
- Turing reducibilities

Theory of Computation

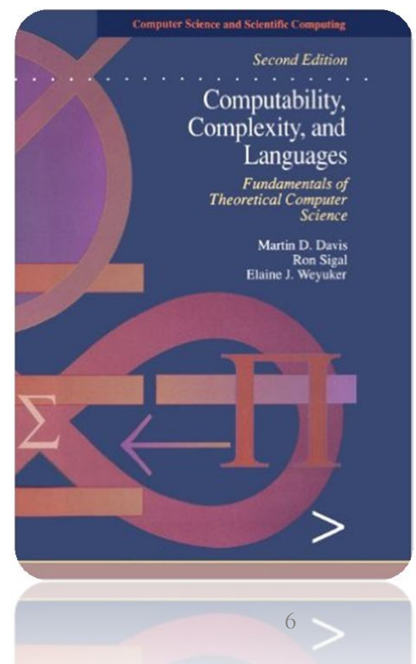
- Church-Turing thesis, Turing machine & its variations
- exploring the limits of algorithmic solvability
- reducibility as a key method to prove unsolvability
- recursive/partial recursive functions
- decidability in terms of recursion
- arriving at Turing machines
- decidability of logical theories
- Turing reducibilities

[S12] Sipser, Michael. **Introduction to the Theory of Computation**, 3rd edition. Cengage Learning, 2012. (Chapters 3 to 6)



Theory of Computation

- Church-Turing thesis, Turing machine & its variations
- exploring the limits of algorithmic solvability
- reducibility as a key method to prove unsolvability
- **recursive/partial recursive functions**
- **decidability in terms of recursion**
- **arriving at Turing machines**
- decidability of logical theories
- Turing reducibilities



[DSW94] Davis, Martin, Ron Sigal, and Elaine J. Weyuker. **Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science**. Newnes, 1994. (Chapters 2 to 6)

Theory of Computation

Course note based on *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd edition, authored by Martin Davis, Ron Sigal, and Elaine J. Weyuker.

course note prepared by

Tyng–Ruey Chuang

Institute of Information Science, Academia Sinica

Department of Information Management, National Taiwan University

Week 1, Spring 2010

Textbook

Martin Davis, Ron Sigal, and Elaine J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd edition. February 1994, Morgan Kaufmann. ISBN: 0122063821.

- Written for people who may know programming, but from a mathematical view of the subjects. Enjoyably readable but very rigorous.
- "It is our purpose . . . to provide an introduction to the various aspects of theoretical computer science for undergraduate and graduate students that is sufficiently comprehensive that . . . research papers will become accessible to our readers." (the authors)
- We will cover just one half of the materials in the book.

Outline of Today's Lecture

- Review some preliminary materials.
- Define an abstract programming language $\mathscr{S}$ that is extremely simple.
- Write some programs in $\mathscr{S}$.

Cartesian Product

- If $S_1, S_2, \ldots, S_n$ are given sets, then we write $S_1 \times S_2, \times \cdots \times S_n$ for the set of all n-tuples $(a_1, a_2, \ldots, a_n)$ such that $a_1 \in S_1, a_2 \in S_2, \ldots, a_n \in S_n$.
- $S_1 \times S_2, \times \cdots \times S_n$ is called the *Cartesian product* of $S_1, S_2, \ldots, S_n$.
- In case $S_1 = S_2 = \cdots = S_n = S$ we write S^n for the Cartesian product $S_1 \times S_2, \times \cdots \times S_n$.

Functions

- A function f is a set whose members are ordered pairs (i.e., 2-tuples) and has the special property

  $$(a, b) \in f \text{ and } (a, c) \in f \quad \text{implies} \quad b = c.$$

  We write $f(a) = b$ to mean that $(a, b) \in f$.

Functions

- A function f is a set whose members are ordered pairs (i.e., 2-tuples) and has the special property

$$(a, b) \in f \text{ and } (a, c) \in f \quad \text{implies} \quad b = c.$$

  We write $f(a) = b$ to mean that $(a, b) \in f$.
- The set of all a such that $(a, b) \in f$ for some b is called the *domain* of f. The set of all $f(a)$ for a in the domain of f is called the *range* of f.

Functions

- A function f is a set whose members are ordered pairs (i.e., 2-tuples) and has the special property

$$(a, b) \in f \text{ and } (a, c) \in f \quad \text{implies} \quad b = c.$$

  We write $f(a) = b$ to mean that $(a, b) \in f$.
- The set of all a such that $(a, b) \in f$ for some b is called the *domain* of f. The set of all $f(a)$ for a in the domain of f is called the *range* of f.
- A *partial function* on a set S is a function whose domain is a subset of S. If a partial function on S has the domain S, then it is called a *total function*.

Functions

- A function f is a set whose members are ordered pairs (i.e., 2-tuples) and has the special property

$$(a, b) \in f \text{ and } (a, c) \in f \quad \text{implies} \quad b = c.$$

  We write $f(a) = b$ to mean that $(a, b) \in f$.
- The set of all a such that $(a, b) \in f$ for some b is called the *domain* of f. The set of all $f(a)$ for a in the domain of f is called the *range* of f.
- A *partial function* on a set S is a function whose domain is a subset of S. If a partial function on S has the domain S, then it is called a *total function*.
- We write $f(a)\downarrow$ and say that $f(a)$ is *defined* if a is in the domain of f; if a is not in the domain of f, we write $f(a)\uparrow$ and say that $f(a)$ is *undefined*.

Examples of Functions

- Let f be the set of ordered pairs (n, n^2) for $n \in N$. Then, for each $n \in N$, $f(n) = n^2$. The domain of f is N. The range of f is the set of perfect squares. f is a total function.

Examples of Functions

- Let f be the set of ordered pairs (n, n^2) for $n \in N$. Then, for each $n \in N$, $f(n) = n^2$. The domain of f is N. The range of f is the set of perfect squares. f is a total function.
- Assuming N is our universe, an example of a partial function on N is given by $g(n) = \sqrt{n}$. The domain of g is the set of perfect squares. The range of g is N. g is not a total function.

Examples of Functions

- Let f be the set of ordered pairs (n, n^2) for $n \in N$. Then, for each $n \in N$, $f(n) = n^2$. The domain of f is N. The range of f is the set of perfect squares. f is a total function.
- Assuming N is our universe, an example of a partial function on N is given by $g(n) = \sqrt{n}$. The domain of g is the set of perfect squares. The range of g is N. g is not a total function.
- For a partial function f on a Cartesian product $S_1 \times S_2, \times \cdots \times S_n$, we write $f(a_1, \ldots, a_n)$ rather than $f((a_1, \ldots, a_n))$.

Examples of Functions

- Let f be the set of ordered pairs (n, n^2) for $n \in N$. Then, for each $n \in N$, $f(n) = n^2$. The domain of f is N. The range of f is the set of perfect squares. f is a total function.
- Assuming N is our universe, an example of a partial function on N is given by $g(n) = \sqrt{n}$. The domain of g is the set of perfect squares. The range of g is N. g is not a total function.
- For a partial function f on a Cartesian product $S_1 \times S_2, \times \cdots \times S_n$, we write $f(a_1, \ldots, a_n)$ rather than $f((a_1, \ldots, a_n))$.
- A partial function f on a set S^n is called an *n-ary* partial function on S, or a function of n variables on S. We use *unary* and *binary* for 1-ary and 2-ary, respectively.

Predicate

A *predicate*, or a *Boolean-valued function*, on a set S is a *total* function P on S such that for each $a \in S$, either

$$P(a) = \text{TRUE} \quad \text{or} \quad P(a) = \text{FALSE}$$

We also identify the truth value TRUE with number 1 and the truth value FALSE with number 0.

Logic Connectives

The three *logic connectives*, or *propositional connectives*, $\sim, \vee, \&$ are defined by the two tables below.

p	$\sim p$
0	1
1	0

p	q	$p \,\&\, q$	$p \vee q$
1	1	1	1
0	1	0	1
1	0	0	1
0	0	0	0

Characteristic Function

Given a predicate P on a set S, there is a corresponding subset R of S consisting of all elements $a \in S$ for which $P(a) = 1$. We write

$$R = \{a \in S \mid P(a)\}.$$

Characteristic Function

Given a predicate P on a set S, there is a corresponding subset R of S consisting of all elements $a \in S$ for which $P(a) = 1$. We write

$$R = \{a \in S \mid P(a)\}.$$

Conversely, given a subset R of a given set S, the expression $x \in R$ defines a predicate P on S:

$$P(x) = \begin{cases} 1 & \text{if } x \in R \\ 0 & \text{if } x \notin R. \end{cases}$$

The predicate P is called the *characteristic function* of the set R.

Characteristic Function

Given a predicate P on a set S, there is a corresponding subset R of S consisting of all elements $a \in S$ for which $P(a) = 1$. We write

$$R = \{a \in S \mid P(a)\}.$$

Conversely, given a subset R of a given set S, the expression $x \in R$ defines a predicate P on S:

$$P(x) = \begin{cases} 1 & \text{if } x \in R \\ 0 & \text{if } x \notin R. \end{cases}$$

The predicate P is called the *characteristic function* of the set R. Note the easy translations between the two notations:

$$\begin{aligned} \{x \in S \mid P(x) \,\&\, Q(x)\} &= \{x \in S \mid P(x)\} \cap \{x \in S \mid Q(x)\}, \\ \{x \in S \mid P(x) \vee Q(x)\} &= \{x \in S \mid P(x)\} \cup \{x \in S \mid Q(x)\}, \\ \{x \in S \mid \sim P(x)\} &= S - \{x \in S \mid P(x)\}. \end{aligned}$$

Bounded Existential Quantifier

Let $P(t, x_1, \ldots, x_n)$ be a $(n+1)$-ary predicate. Let predicate $Q(y, x_1, \ldots, x_n)$ be defined by

$$\begin{array}{rcl} Q(y, x_1, \ldots, x_n) & = & P(0, x_1, \ldots, x_n) \\ & \vee & P(1, x_1, \ldots, x_n) \\ & \vee & \ldots \\ & \vee & P(y, x_1, \ldots, x_n) \end{array}$$

Bounded Existential Quantifier

Let $P(t, x_1, \ldots, x_n)$ be a $(n+1)$-ary predicate. Let predicate $Q(y, x_1, \ldots, x_n)$ be defined by

$$\begin{array}{rcl} Q(y, x_1, \ldots, x_n) & = & P(0, x_1, \ldots, x_n) \\ & \vee & P(1, x_1, \ldots, x_n) \\ & \vee & \ldots \\ & \vee & P(y, x_1, \ldots, x_n) \end{array}$$

That is, $Q(y, x_1, \ldots, x_n)$ is true if there is a value $t \leq y$ such that $P(t, x_1, \ldots, x_n)$ is true. We write this predicate Q as

$$(\exists t)_{\leq y} P(t, x_1, \ldots, x_n)$$

"$(\exists t)_{\leq y}$" is called a *bounded existential quantifier*.

Bounded Universal Quantifier

Let $P(t, x_1, \ldots, x_n)$ be a $(n+1)$-ary predicate. Let predicate $Q(y, x_1, \ldots, x_n)$ be defined by

$$\begin{aligned} Q(y, x_1, \ldots, x_n) &= P(0, x_1, \ldots, x_n) \\ &\;\& \;\; P(1, x_1, \ldots, x_n) \\ &\;\& \;\; \ldots \\ &\;\& \;\; P(y, x_1, \ldots, x_n) \end{aligned}$$

Bounded Universal Quantifier

Let $P(t, x_1, \ldots, x_n)$ be a $(n+1)$-ary predicate. Let predicate $Q(y, x_1, \ldots, x_n)$ be defined by

$$\begin{array}{rcl} Q(y, x_1, \ldots, x_n) & = & P(0, x_1, \ldots, x_n) \\ & \& & P(1, x_1, \ldots, x_n) \\ & \& & \ldots \\ & \& & P(y, x_1, \ldots, x_n) \end{array}$$

That is, $Q(y, x_1, \ldots, x_n)$ is true if for all value $t \leq y$ such that $P(t, x_1, \ldots, x_n)$ is true. We write this predicate Q as

$$(\forall t)_{\leq y} P(t, x_1, \ldots, x_n)$$

"$(\forall t)_{\leq y}$" is called a *bounded universal quantifier*.

The Programming Language $\mathscr{S}$

- Values: natural numbers only, but of unlimited precision.

The Programming Language $\mathscr{S}$

- Values: natural numbers only, but of unlimited precision.
- Variables:
  - Input variables $X_1, X_1, X_3, \ldots$
  - An output variable Y
  - Local variables $Z_1, Z_1, Z_3, \ldots$

The Programming Language $\mathscr{S}$

- Values: natural numbers only, but of unlimited precision.
- Variables:
  - Input variables $X_1, X_1, X_3, \ldots$
  - An output variable Y
  - Local variables $Z_1, Z_1, Z_3, \ldots$
- Instructions:

$V \leftarrow V + 1$	Increase by 1 the value of the variable V.
$V \leftarrow V - 1$	If the value of V is 0, leave it unchanged; otherwise decrease by 1 the value of 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 $\mathscr{S}$

- Values: natural numbers only, but of unlimited precision.
- Variables:
  - Input variables $X_1, X_1, X_3, \ldots$
  - An output variable Y
  - Local variables $Z_1, Z_1, Z_3, \ldots$
- Instructions:

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

- Labels: $A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, A_3, \ldots$
- Exit label: E.

The Programming Language $\mathscr{S}$

- Values: natural numbers only, but of unlimited precision.
- Variables:
  - Input variables $X_1, X_1, X_3, \ldots$
  - An output variable Y
  - Local variables $Z_1, Z_1, Z_3, \ldots$
- Instructions:

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

- Labels: $A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, A_3, \ldots$
- Exit label: E.
- All variables and labels are in the global scope.

Programming in $\mathscr{S}$

- A program is a list (i.e., a finite sequence) of instructions.
- *The output variable* Y *and the local variables* Z_i *initially have the value* 0.
- A program halts when there is no more instruction to execute.
- A program also halts if an instruction labeled L is to be executed, but there is no instruction with that label.

Programming in $\mathscr{S}$

- A program is a list (i.e., a finite sequence) of instructions.
- *The output variable* Y *and the local variables* Z_i *initially have the value* 0.
- A program halts when there is no more instruction to execute.
- A program also halts if an instruction labeled L is to be executed, but there is no instruction with that label.
- What does this program do?

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

A Bug?

- What does this program do?

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

A Bug?

- What does this program do?

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

- The above program *computes* the function

$$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ x & \text{otherwise.} \end{cases}$$

A Program That Computes $f(x) = x$

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

A Program That Computes $f(x) = x$

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

- What does Z actually do?

A Program That Computes $f(x) = x$

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

- What does Z actually do?
- What does the following do?

$$
\begin{array}{l}
Z \leftarrow Z + 1 \\
\text{IF } Z \neq 0 \text{ GOTO } L
\end{array}
$$

A Program That Computes $f(x) = x$

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

- What does Z actually do?
- What does the following do?

$$
\begin{array}{l}
Z \leftarrow Z + 1 \\
\text{IF } Z \neq 0 \text{ GOTO } L
\end{array}
$$

- That is an unconditional goto!

$$\text{GOTO } L$$

A *Macro* for Unconditional GOTO

- Before macro expansion:

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

A *Macro* for Unconditional GOTO

- Before macro expansion:

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

- After macro expansion:

$[A]$	IF $X \neq 0$ GOTO B
	$Z_1 \leftarrow Z_1 + 1$
	IF $Z_1 \neq 0$ GOTO E
$[B]$	$X \leftarrow X - 1$
	$Y \leftarrow Y + 1$
	$Z_2 \leftarrow Z_2 + 1$
	IF $Z_2 \neq 0$ GOTO A

A *Macro* for Unconditional GOTO

- Before macro expansion:

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

- After macro expansion:

  $[A]$ IF $X \neq 0$ GOTO B
  $Z_1 \leftarrow Z_1 + 1$
  IF $Z_1 \neq 0$ GOTO E
  $[B]$ $X \leftarrow X - 1$
  $Y \leftarrow Y + 1$
  $Z_2 \leftarrow Z_2 + 1$
  IF $Z_2 \neq 0$ GOTO A

- *Fresh local variables are always used during macro expansions.*

Copy The Value of Variable X to Variable Y

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

Copy The Value of Variable X to Variable Y

- $[A]$ IF $X \neq 0$ GOTO B
  GOTO E
  $[B]$ $X \leftarrow X - 1$
  $Y \leftarrow Y + 1$
  GOTO A
- Anything wrong?

Copy The Value of Variable X to Variable Y

- $$\begin{array}{ll} [A] & \text{IF } X \neq 0 \text{ GOTO } B \\ & \text{GOTO } E \\ [B] & X \leftarrow X - 1 \\ & Y \leftarrow Y + 1 \\ & \text{GOTO } A \end{array}$$
- Anything wrong?
- The value of X is "destroyed" while copied to Y!

Copy The Value of Variable X to Variable Y, Continued

- $[A]$ IF $X \neq 0$ GOTO B
  GOTO C
  $[B]$ $X \leftarrow X - 1$
  $Y \leftarrow Y + 1$
  $Z \leftarrow Z + 1$
  GOTO A
  $[C]$ IF $Z \neq 0$ GOTO D
  GOTO E
  $[D]$ $Z \leftarrow Z - 1$
  $X \leftarrow X + 1$
  GOTO C

Copy The Value of Variable X to Variable Y, Continued

- $[A]$ IF $X \neq 0$ GOTO B
  GOTO C
  $[B]$ $X \leftarrow X - 1$
  $Y \leftarrow Y + 1$
  $Z \leftarrow Z + 1$
  GOTO A
  $[C]$ IF $Z \neq 0$ GOTO D
  GOTO E
  $[D]$ $Z \leftarrow Z - 1$
  $X \leftarrow X + 1$
  GOTO C
- Anything wrong?

Copy The Value of Variable X to Variable Y, Continued

- [A] IF $X \neq 0$ GOTO B
  GOTO C
  [B] $X \leftarrow X - 1$
  $Y \leftarrow Y + 1$
  $Z \leftarrow Z + 1$
  GOTO A
  [C] IF $Z \neq 0$ GOTO D
  GOTO E
  [D] $Z \leftarrow Z - 1$
  $X \leftarrow X + 1$
  GOTO C
- Anything wrong?
- This program is correct only when Y and Z are initialized to the value 0. It cannot be used as a macro.

A Macro for $V \leftarrow V'$

- $$\begin{array}{ll}
 & V \leftarrow 0 \\
[A] & \text{IF } V' \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & V \leftarrow V' - 1 \\
 & V \leftarrow V + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & V' \leftarrow V' + 1 \\
 & \text{GOTO } C
\end{array}$$

A Macro for $V \leftarrow V'$

- $$
\begin{array}{ll}
 & V \leftarrow 0 \\
[A] & \text{IF } V' \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & V \leftarrow V' - 1 \\
 & V \leftarrow V + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & V' \leftarrow V' + 1 \\
 & \text{GOTO } C
\end{array}
$$
- Anything wrong?

A Macro for $V \leftarrow V'$

- $$\begin{array}{ll} & V \leftarrow 0 \\ [A] & \text{IF } V' \neq 0 \text{ GOTO } B \\ & \text{GOTO } C \\ [B] & V \leftarrow V' - 1 \\ & V \leftarrow V + 1 \\ & Z \leftarrow Z + 1 \\ & \text{GOTO } A \\ [C] & \text{IF } Z \neq 0 \text{ GOTO } D \\ & \text{GOTO } E \\ [D] & Z \leftarrow Z - 1 \\ & V' \leftarrow V' + 1 \\ & \text{GOTO } C \end{array}$$
- Anything wrong?
- $V \leftarrow 0$ is not an instruction in $\mathscr{S}$.

A Macro for $V \leftarrow 0$

A Macro for $V \leftarrow 0$

$$
\begin{array}{ll}
[L] & V \leftarrow V - 1 \\
 & \text{IF } V \neq 0 \text{ GOTO } L
\end{array}
$$

A Program That Computes $f(x_1, x_2) = x_1 + x_2$

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[B] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z \leftarrow Z - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{GOTO } B
\end{array}
$$

Note that Z is used to preserve the value of X_2 so that it will not be destroyed during the computation.