Theory of Computation

Prof. Michael Mascagni

Florida State University
Department of Computer Science

This course aims to cover ...

▶ the development of computability theory using an extremely simple abstract programming language,

▶ the various different formulations of computability and their equivalence

By the end of this course, you should be able to . . .

- ▶ appreciate the existence of universal digital computers,
- ▶ understand there are well-defined functions that cannot be computed even by the universal computers,
- ▶ know that certain problems are truly harder than others,
- ▶ use various formalized computation models to solve your problems, and
- ▶ show that some problems are just too difficult for the models at hand.

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 Lecture

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

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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 $S_1 \times S_2, \times \cdots \times S_n$.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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*.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, n-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, n-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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:

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

Preliminaries (1)
Programs and Computable Functions (2)

Sets, n-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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

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

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*.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, n-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

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

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

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*.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

Proof by Contradiction

In a *proof by contradiction*, we begin by assuming the assertion we wish to prove is *false*. We then derive a contradiction based on this (faulty) assumption along with (faultless) logical reasoning. We then conclude that the original assertion must be true.

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

Proof by Contradiction: Example

Prove that the equation $2 = (m/n)^2$ has no solution $m, n \in N$.

Proof. Assume $2 = (m/n)^2$ *has* a solution $m, n \in N$. Then it must also have a solution where not both m and n are even. This is so because we can repeatedly "cancel" 2 from m and n until at least one of them becomes *odd*, and still have the two "reduced" numbers as a solution.

However, the equation $2 = (m/n)^2$ can be rewritten as $m^2 = 2n^2$ which shows that m must be even. Let $m = 2k$, then $m^2 = (2k)^2 = 4k^2$. But this implies $n^2 = 2k^2$. Thus n is even. Now both m and n are even, which is a contradiction.

We conclude that $2 = (m/n)^2$ has no solution $m, n \in N$. □

Preliminaries (1)
Programs and Computable Functions (2)

Sets, *n*-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

Mathematical Induction

Given a predicate $P(x)$, and the assertion "$P(n)$ is true for all $n \in N$", we can use mathematical induction to try to establish this assertion. One proceeds by proving a pair of auxiliary statements about $P(x)$, namely,

$P(0)$

and

For all $n \in N$, $P(n)$ implies $P(n + 1)$

In the second statement above, $P(n)$ is called an induction hypothesis. If both statements above are proved to be true, one then concludes that

For all $n \in N$, $P(n)$

Preliminaries (1)
Programs and Computable Functions (2)

Sets, n-tuples, and functions (1.1, 1.2)
Predicates (1.4)
Quantifiers (1.5)
Proofs (1.6, 1.7)

Mathematical Induction: Example

Prove that for all $n \in N$, $\sum_{i=0}^{n}(2i + 1) = (n + 1)^2$.

Proof. For $n = 0$, then $\sum_{i=0}^{0}(2i + 1) = 1 = (0 + 1)^2$, which is true.
It remains to show that for all $n \in N$, if $\sum_{i=0}^{n}(2i + 1) = (n + 1)^2$ is
true, then $\sum_{i=0}^{n+1}(2i + 1) = (n + 2)^2$ is also true.
We expand $\sum_{i=0}^{n+1}(2i + 1)$ by its definition,

$$
\begin{aligned}
\sum_{i=0}^{n+1}(2i + 1) &= \sum_{i=0}^{n}(2i + 1) + 2(n + 1) + 1 \\
&= (n + 1)^2 + 2(n + 1) + 1 \quad \text{(by induction hypothesis)} \\
&= (n + 2)^2.
\end{aligned}
$$

We conclude that for all $n \in N$, $\sum_{i=0}^{n}(2i + 1) = (n + 1)^2$. $\qquad \square$

The Programming Language $\mathscr{S}$

- ▶ Values: natural numbers only, but of unlimited precision.
- ▶ Variables:
  - ▶ Input variables $X_1, X_2, X_3, \ldots$
  - ▶ An output variable Y
  - ▶ Local variables $Z_1, Z_2, Z_3, \ldots$
- ▶ 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.
▶ What does this program do?

$$[A] \quad X \leftarrow X - 1$$
$$Y \leftarrow Y + 1$$
$$\text{IF } X \neq 0 \text{ GOTO } A$$

A Bug?

▶ What does this program do?

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

▶ The above program *computes* the function

$$f(x) = \left\{ \begin{array}{ll} 1 & \text{if } x = 0 \\ x & \text{otherwise}. \end{array} \right.$$

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

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

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

       $Z \leftarrow Z + 1$
       IF $Z \neq 0$ GOTO L

▶ That is an unconditional goto!
       GOTO L

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
- 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
- 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 $V \leftarrow V'$

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

- Anything wrong?
- $V \leftarrow 0$ is not an instruction in $\mathscr{S}$.

A Macro for $V \leftarrow 0$

$[L]$ $V \leftarrow V - 1$
       IF $V \neq 0$ GOTO L

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

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

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

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

-       $Z_2 \leftarrow X_2$
  - [B] IF $Z_2 \neq 0$ GOTO A
  -       GOTO E
  - [A] $Z_2 \leftarrow Z_2 - 1$
  -       $Z_1 \leftarrow X_1 + Y$
  -       $Y \leftarrow Z_1$
  -       GOTO B
- OK!

A Shorter Program That Computes $f(x_1, x_2) = x_1 \cdot x_2$?

▶ $Z_2 \leftarrow X_2$
　[B]   IF $Z_2 \neq 0$ GOTO A
　　　 GOTO E
　[A] $Z_2 \leftarrow Z_2 - 1$
　　　 $Y \leftarrow X_1 + Y$
　　　 GOTO B
▶ *NO GOOD!*
▶ Why?

- The macro for $f(x_1, x_2) = x_1 + x_2$

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

- Macro expanding $Y \leftarrow X_1 + Y$:

$$Y \leftarrow X_1$$
$$Z \leftarrow Y$$
$[B]$ IF $Z \neq 0$ GOTO A
GOTO E
$[A]$ $Z \leftarrow Z - 1$
$Y \leftarrow Y + 1$
GOTO B

- The above actually computes $f(x_1, x_2) = 2 \cdot x_1$

A Program That Computes $f(x_1, x_2) = x_1 \cdot x_2$, Revisited

- ▶ Need to macro expand $Z_1 \leftarrow X_1 + Y$.
- ▶ After macro expansion:

$$Z_2 \leftarrow X_2$$
[B]   IF $Z_2 \neq 0$ GOTO A
      GOTO E
[A] $Z_2 \leftarrow Z_2 - 1$
      $Z_1 \leftarrow X_1$
      $Z_3 \leftarrow Y$
[B_2]   IF $Z_3 \neq 0$ GOTO A_2
      GOTO E_2
[A_2] $Z_3 \leftarrow Z_3 - 1$
      $Z_1 \leftarrow Z_1 + 1$
      GOTO B_2
[E_2] $Y \leftarrow Z_1$
      GOTO B

Note on The Macro Expansion

▶ The output variable Y in the macro $f(x_1, x_2) = x_1 + x_2$ is now fresh variable Z_1 in the expanded form.

▶ The local variable Z in the macro $f(x_1, x_2) = x_1 + x_2$ is now fresh variable Z_3 in the expanded form (as variables Z_1 and Z_2 are already used).

▶ Fresh labels A_2, B_2, and E_2 are used in the expanded form (as the original labels A, B, and E are already used).

▶ The instruction GOTO E_2 only terminates the addition. The computation must continue to place following the addition. Hence, the instruction immediately following the addition is labeled E_2.

▶ *Unlimited supply of fresh local variables and local labels!*

▶ More about macro expansion next week.

A Final Example

▶ What does this program compute?

$$Y \leftarrow X_1$$
$$Z \leftarrow X_2$$
[C]   IF $Z \neq 0$ GOTO A
       GOTO E
[A]   IF $Y \neq 0$ GOTO B
       GOTO A
[B] $Y \leftarrow Y - 1$
       $Z \leftarrow Z - 1$
       GOTO C

▶ If we begin with $X_1 = 5$ and $X_2 = 2$, ...
▶ If we begin with $X_1 = 2$ and $X_2 = 5$, ...
▶ This program computes the following *partial function*

$$g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ \uparrow & \text{if } x_1 < x_2 \end{cases}$$