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

Week 1, Spring 2008

About This Course Note

- It is prepared for the course *Theory of Computation* taught at the National Taiwan University in Spring 2008.
- It follows very closely the book *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd edition, by Martin Davis, Ron Sigal, and Elaine J. Weyuker. Morgan Kaufmann Publishers. ISBN: 0-12-206382-1.
- It is available from Tyng-Ruey Chuang's web site:

  `http://www.iis.sinica.edu.tw/~trc/`

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,
- the expressiveness and limitation of various kinds of automata and formal languages, and
- the basics of the theory of computational complexity.

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.

Schedule (1/2)

02/20 Preliminaries; A Programming Language. (1.1–1.7; 2.1–2.2)

02/27 Computable Functions; Primitive Recursive Functions. (2.3–2.5; 3.1–3.4)

03/05 Coding Programs by Numbers. (3.5–3.8; 4.1)

03/12 The Halting Problem; Universality. (4.2–4.3)

03/19 Recursively Enumerable Sets. (4.4–4.5)

03/26 Diagonalization and Reducibility. (4.6–4.8)

04/02 A Computable Function That Is Not Primitive Recursive. (4.9)

04/09 Turing Machines. (6.1–6.4)

04/16 *mid-term examination*

Schedule (2/2)

04/23 Nondeterministic Turing Machines; Semi-Thue Processes. (6.5–6.5; 7.1–7.2)

04/30 Post's Correspondence Problem. Grammars. (7.2–7.6)

05/07 Regular Languages, Part 1. (9.1–9.4)

05/14 Regular Languages, Part 2. (9.5–9.7)

05/21 Context-Free Languages, Part 1. (10.1–10.4)

05/28 Context-Free Languages, Part 2. (10.5–10.9)

06/04 Context-Sensitive Languages. (11.1–11.3)

06/11 Polynomial-Time Computability. (15.1–15.4)

06/18 *final examination*

Outline of Today's Lecture

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

1 Preliminaries (1)

1.1 Sets and n-tuples (1.1)

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

1.2 Functions (1.2)

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.

- 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.

2 Programs and Computable Functions (2)

2.1 A Programming Language (2.1)

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.

2.2 Some Examples of Programs (2.2)

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?

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

- The above program *computes* the function

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

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

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

- 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 *Macro* for Unconditional GOTO

- Before macro expansion:

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

- After macro expansion:

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & Z_1 \leftarrow Z_1 + 1 \\
 & \text{IF } Z_1 \neq 0 \text{ GOTO } E \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z_2 \leftarrow Z_2 + 1 \\
 & \text{IF } Z_2 \neq 0 \text{ GOTO } A
\end{array}
$$

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

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

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

- 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 $V \leftarrow V'$

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

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

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.

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

- $$
\begin{array}{ll}
 & Z_2 \leftarrow X_2 \\
[B] & \text{IF } Z_2 \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z_2 \leftarrow Z_2 - 1 \\
 & Z_1 \leftarrow X_1 + Y \\
 & Y \leftarrow Z_1 \\
 & \text{GOTO } B
\end{array}
$$
- OK!

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

- $$
\begin{array}{ll}
 & Z_2 \leftarrow X_2 \\
[B] & \text{IF } Z_2 \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z_2 \leftarrow Z_2 - 1 \\
 & Y \leftarrow X_1 + Y \\
 & \text{GOTO } B
\end{array}
$$
- *NO GOOD!*

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

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

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

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

- 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:

$$
\begin{array}{ll}
 & Z_2 \leftarrow X_2 \\
[B] & \text{IF } Z_2 \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z_2 \leftarrow Z_2 - 1 \\
 & Z_1 \leftarrow X_1 \\
 & Z_3 \leftarrow Y \\
[B_2] & \text{IF } Z_3 \neq 0 \text{ GOTO } A_2 \\
 & \text{GOTO } E_2 \\
[A_2] & Z_3 \leftarrow Z_3 - 1 \\
 & Z_1 \leftarrow Z_1 + 1 \\
 & \text{GOTO } B_2 \\
[E_2] & Y \leftarrow Z_1 \\
 & \text{GOTO } B
\end{array}
$$

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?

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

- 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*

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