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 2, Spring 2010

About This Course Note

- It is prepared for the course *Theory of Computation* taught at the National Taiwan University in Spring 2010.
- 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/`

  and released under a Creative Commons "Attribution-ShareAlike 3.0 Taiwan" license:

  `http://creativecommons.org/licenses/by-sa/3.0/tw/`

Syntax of Language $\mathscr{S}$

- Variables:
  - Input variables: $X_1, X_2, X_3, \ldots$
  - Output variable: Y
  - Local variables: $Z_1, Z_2, Z_3, \ldots$
- Labels: $A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, A_3, \ldots$
- A statement is one of the following:
  - $V \leftarrow V + 1$
  - $V \leftarrow V - 1$
  - $V \leftarrow V$
  - IF $V \neq 0$ GOTO L

  where V may be any variable and L may be any label.

Syntax of Language $\mathscr{S}$

- Variables:
  - Input variables: $X_1, X_2, X_3, \ldots$
  - Output variable: Y
  - Local variables: $Z_1, Z_2, Z_3, \ldots$
- Labels: $A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, A_3, \ldots$
- A statement is one of the following:
  - $V \leftarrow V + 1$
  - $V \leftarrow V - 1$
  - $V \leftarrow V$
  - IF $V \neq 0$ GOTO L

  where V may be any variable and L may be any label.

Note: X_1 is a shorthand for X, Z_1 is a shorthand for Z, and A is a shorthand for A_1, etc.
$V \leftarrow V$ are harmless "dummy" commands (more on these statements later).

Program

- An *instruction* is either a statement or $[L]$ followed by a statement.
- A *program* is a list (i.e., a finite sequence) of instruction. The length of this list is called the *length* of the program. The *empty* program is of length 0.

State

- A *state of a program* $\mathscr{P}$ is a list of equations of the form

$$V = m$$

where V is a variable and m is a number, including an equation for each variable that occurs in $\mathscr{P}$ and including no two equations with the same variable.

- Let σ be a state of $\mathscr{P}$ and let V be a variable that occurs in σ. The *value* of V at σ is the (unique) number q such that the equation $V = q$ is one of the equations making up σ.

State

- A *state of a program* $\mathscr{P}$ is a list of equations of the form

$$V = m$$

  where V is a variable and m is a number, including an equation for each variable that occurs in $\mathscr{P}$ and including no two equations with the same variable.
- Let σ be a state of $\mathscr{P}$ and let V be a variable that occurs in σ. The *value* of V at σ is the (unique) number q such that the equation $V = q$ is one of the equations making up σ.

Note: An *number* is a nonnegative integer.

A State of A Program: Examples

$[A]$ 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

Given program $\mathscr{P}$ above, each of the following is a state of $\mathscr{P}$:

- $X = 4, Y = 3, Z = 3$
- $X_1 = 4, X_2 = 5, Y = 4, Z = 4$

but each of the following is *not* a state of $\mathscr{P}$

- $X = 3, Z = 3$
- $X = 3, X = 4, Y = 2, Z = 2$

Snapshot

Let $\mathscr{P}$ be a program of length n. Then,

- A *snapshot*, or *instantaneous description*, of program $\mathscr{P}$ is a pair (i, σ) where $1 \leq i \leq n+1$, and σ is a state of $\mathscr{P}$.
- The *value* of a variable V at a snapshot (i, σ) just means the value of V at σ.
- Intuitively the number i indicates that it is the ith instruction which is about to be executed; $i = n+1$ corresponds to a "stop" instruction. A snapshot with $i = n+1$ is called terminal.

The Successor of A Snapshot

Let (i, σ) be a nonterminal snapshot of program $\mathscr{P}$, then its *successor* (j, τ) will depend on the ith instruction of $\mathscr{P}$. If the ith instruction is

$\boxed{V \leftarrow V + 1}$ then $j = i + 1$, and τ is σ with equation $V = m$ replaced by $V = m + 1$;

$\boxed{V \leftarrow V - 1}$ then $j = i + 1$, and τ is σ with equation $V = m$ replaced by $V = m - 1$;

$\boxed{V \leftarrow V}$ then $j = i + 1$, and $\tau = \sigma$;

$\boxed{\text{IF } V \neq 0 \text{ GOTO } L}$ then $j = i + 1$, and $\tau = \sigma$ if the value of V at σ is 0; otherwise $\tau = \sigma$ and j is the *least number* such that the jth instruction of $\mathscr{P}$ is labeled L (in case no instruction in $\mathscr{P}$ is labeled L, let $j = n + 1$).

The Successor of A Snapshot: Examples

$[A]$ 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

Given program $\mathscr{P}$ above, then

- the successor of $(1, \{X = 4, Y = 0, Z = 0\})$ is $(4, \{X = 4, Y = 0, Z = 0\})$;
- the successor of $(2, \{X = 4, Y = 0, Z = 0\})$ is $(3, \{X = 4, Y = 0, Z = 1\})$;
- the successor of $(7, \{X = 4, Y = 0, Z = 0\})$ is $(8, \{X = 4, Y = 0, Z = 0\})$ which is a terminal snapshot.

Computation

A *computation* of a program $\mathscr{P}$ is defined to be a sequence (i.e., a list) $s_1, s_2, \ldots, s_k$ of snapshots of $\mathscr{P}$ such that s_{i+1} is the successor of s_i for $i = 1, 2, \ldots, k-1$ and s_k is terminal.

Computation from An Initial State

Let $\mathscr{P}$ be a program, and let $r_1, r_2, \ldots, r_m$ be m given numbers. The state σ of $\mathscr{P}$ is defined to consist of the equations

$$X_1 = r_1,\ X_2 = r_2,\ \ \ldots,\ X_m = r_m,\ Y = 0$$

together with the equation $V = 0$ for each variable V in $\mathscr{P}$ other than $X_1, X_2, \ldots, X_m, Y$. This state is called the *initial state*, and the snapshot $(1, \sigma)$ *the initial snapshot*.

Computation from An Initial State

Let $\mathscr{P}$ be a program, and let $r_1, r_2, \ldots, r_m$ be m given numbers. The state σ of $\mathscr{P}$ is defined to consist of the equations

$$X_1 = r_1,\ X_2 = r_2,\ \ \ldots,\ X_m = r_m,\ Y = 0$$

together with the equation $V = 0$ for each variable V in $\mathscr{P}$ other than $X_1, X_2, \ldots, X_m, Y$. This state is called the *initial state*, and the snapshot $(1, \sigma)$ *the initial snapshot*.
Starting from the initial snapshot $s_1 = (1, \sigma)$, there can be either

- a computation $s_1, s_2, \ldots, s_k$ of $\mathscr{P}$, or
- no such computation (i.e., there is an *infinite* sequence $s_1, s_2, s_3, \ldots$ where each s_{k+1} is the successor of s_k).

Computation from An Initial State

Let $\mathscr{P}$ be a program, and let $r_1, r_2, \ldots, r_m$ be m given numbers. The state σ of $\mathscr{P}$ is defined to consist of the equations

$$X_1 = r_1,\ X_2 = r_2,\ \ \ldots,\ X_m = r_m,\ Y = 0$$

together with the equation $V = 0$ for each variable V in $\mathscr{P}$ other than $X_1, X_2, \ldots, X_m, Y$. This state is called the *initial state*, and the snapshot $(1, \sigma)$ *the initial snapshot*.
Starting from the initial snapshot $s_1 = (1, \sigma)$, there can be either

- a computation $s_1, s_2, \ldots, s_k$ of $\mathscr{P}$, or
- no such computation (i.e., there is an *infinite* sequence $s_1, s_2, s_3, \ldots$ where each s_{k+1} is the successor of s_k).

We write $\Psi^{(m)}_{\mathscr{P}}(r_1, r_2, \ldots, r_m)$ for the value of Y at the terminal snapshot. In the case where there is no computation, $\Psi^{(m)}_{\mathscr{P}}(r_1, r_2, \ldots, r_m)$ is undefined.

Function $f(x) = x$, Revisited

$[A]$	IF $X \neq 0$ GOTO B	(1)
	$Z \leftarrow Z + 1$	(2)
	IF $Z \neq 0$ GOTO E	(3)
$[B]$	$X \leftarrow X - 1$	(4)
	$Y \leftarrow Y + 1$	(5)
	$Z \leftarrow Z + 1$	(6)
	IF $Z \neq 0$ GOTO A	(7)

Given program $\mathscr{P}$ above (line numbers added), then

$$\Psi^{(1)}_{\mathscr{P}}(x) = x$$

for all x.

A Computation of Program $\mathscr{P}$

Assuming $r \neq 0$, the snapshots are

$$
\begin{array}{llll}
(1, & \{X = r, & Y = 0, & Z = 0\}), \\
(4, & \{X = r, & Y = 0, & Z = 0\}), \\
(5, & \{X = r - 1, & Y = 0, & Z = 0\}), \\
(6, & \{X = r - 1, & Y = 1, & Z = 0\}), \\
(7, & \{X = r - 1, & Y = 1, & Z = 1\}), \\
(1, & \{X = r - 1, & Y = 1, & Z = 1\}), \\
\ldots, & & & \\
(1, & \{X = 0, & Y = r, & Z = r\}), \\
(2, & \{X = 0, & Y = r, & Z = r\}), \\
(3, & \{X = 0, & Y = r, & Z = r + 1\}), \\
(8, & \{X = 0, & Y = r, & Z = r + 1\})
\end{array}
$$

Partially Computable Functions and Computable Functions

- A given partial function g is said to be *partially computable* if it is computed by some program. That is, g is partially computable if there is a program $\mathscr{P}$ such that

$$g(r_1, \ldots, r_m) = \Psi^{(m)}_{\mathscr{P}}(r_1, \ldots, r_m)$$

for all $t_1, \ldots, r_m$. The above equation is understood to mean not only that both sides agree to the same value when they are defined, but also that when either side is undefined, the other is also undefined.

Partially Computable Functions and Computable Functions

- A given partial function g is said to be *partially computable* if it is computed by some program. That is, g is partially computable if there is a program $\mathscr{P}$ such that

$$g(r_1, \ldots, r_m) = \Psi^{(m)}_{\mathscr{P}}(r_1, \ldots, r_m)$$

  for all $t_1, \ldots, r_m$. The above equation is understood to mean not only that both sides agree to the same value when they are defined, but also that when either side is undefined, the other is also undefined.
- A function is *computable* if it is both *partially computable* and total.

Partially Computable Functions and Computable Functions

- A given partial function g is said to be *partially computable* if it is computed by some program. That is, g is partially computable if there is a program $\mathscr{P}$ such that

$$g(r_1, \ldots, r_m) = \Psi^{(m)}_{\mathscr{P}}(r_1, \ldots, r_m)$$

  for all $t_1, \ldots, r_m$. The above equation is understood to mean not only that both sides agree to the same value when they are defined, but also that when either side is undefined, the other is also undefined.
- A function is *computable* if it is both *partially computable* and total.
- Partially computable functions are also called *partial recursive*, and computable functions are called *recursive*.

A Nowhere Defined Function

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

- For the above program $\mathscr{P}$, $\Psi^{(1)}_{\mathscr{P}}(x)$ is undefined for all x.

A Nowhere Defined Function

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

- For the above program $\mathscr{P}$, $\Psi^{(1)}_{\mathscr{P}}(x)$ is undefined for all x.
- The function

$$f(x) \uparrow, \quad \text{for all } x$$

is partially computable because $f(x) = \Psi^{(1)}_{\mathscr{P}}(x)$.

Computability Theory

- Computability theory (also called recursion theory) studies the class of partially computable functions.

Computability Theory

- Computability theory (also called recursion theory) studies the class of partially computable functions.
- A function can be claimed to be "computable" only when there really is a program of language $\mathscr{S}$ which computes it.

Computability Theory

- Computability theory (also called recursion theory) studies the class of partially computable functions.
- A function can be claimed to be "computable" only when there really is a program of language $\mathscr{S}$ which computes it.
- Is this justified? Isn't the language $\mathscr{S}$ too simplistic and too ad hoc?

Computability Theory

- Computability theory (also called recursion theory) studies the class of partially computable functions.
- A function can be claimed to be "computable" only when there really is a program of language $\mathscr{S}$ which computes it.
- Is this justified? Isn't the language $\mathscr{S}$ too simplistic and too ad hoc?
- More evidence will be developed as we go along! We will show language $\mathscr{S}$ is as powerful as we can get!

Wanted: Macro Expansion without Headache

Let $f(x_1, \ldots, x_n)$ be some partially computable function computed by the program $\mathscr{P}$. How are we able to use macros like

$$W \leftarrow f(V_1, \ldots, V_n)$$

in our programs, where $V_1, \ldots, V_n, W$ can be any variables whatsoever? In particular, W might be one of $V_1, \ldots, V_n$.

A Program Form

- Assume that the variables that occur in $\mathscr{P}$ are all included in the list $Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k$ and that the labels that occur in $\mathscr{P}$ are all included in the list $E, A_1, \ldots, A_l$.
- We also assume that for each instruction of $\mathscr{P}$ of the form

  $$IF\ V \neq 0\ GOTO\ A_i$$

  there is in $\mathscr{P}$ an instruction labeled A_i. In other words, E is the only "exit" label.
- Any program $\mathscr{P}$ can be made to meet the above conditions after minor changes in notation.

Renaming in A Program Form

- We now write

$$\mathscr{P} = \mathscr{P}(Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k; E, A_1, \ldots, A_l)$$

and write

$$\mathscr{Q}_m = \mathscr{P}\ (Z_m, Z_{m+1}, \ldots, Z_{m+n}, Z_{m+n+1}, \ldots, Z_{m+n+k}; E_m, A_{m+1}, \ldots, A_{m+l})$$

for each given value of m.

- The number m is chosen such that all variables and labels in $\mathscr{Q}_m$ are new.

$W \leftarrow f(V_1, \ldots, V_n)$, Macro Expanded

$$
\begin{array}{ll}
 & Z_m \leftarrow 0 \\
 & Z_{m+1} \leftarrow V_1 \\
 & Z_{m+2} \leftarrow V_2 \\
 & \ldots \\
 & Z_{m+n} \leftarrow V_n \\
 & Z_{m+n+1} \leftarrow 0 \\
 & Z_{m+n+2} \leftarrow 0 \\
 & \ldots \\
 & Z_{m+n+k} \leftarrow 0 \\
 & \mathscr{Q}_m \\
[E_m] & W \leftarrow Z_m
\end{array}
$$

Note: If $f(V_1, \ldots, V_n)$ is undefined, the program $\mathscr{Q}_m$ will never terminate.

General Conditional Branch Statement

- Function $P(x_1, \ldots, x_n)$ is a *computable predicate* if it is a computable function returning either 1 (interpreted as TRUE) or 0 (interpreted as FALSE).
- Let $P(x_1, \ldots, x_n)$ be any computable predicate. Then the appropriate macro expansion of

  $$IF\ P(x_1, \ldots, x_n)\ GOTO\ L$$

  is simply

  $$Z \leftarrow P(x_1, \ldots, x_n)$$
  $$IF\ Z \neq 0\ GOTO\ L$$

  where variable Z is new.

Composition

Let f be a function of k variables and let $g_1, \ldots, g_k$ be functions of n variables. Let

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)).$$

The h is said to be obtained from f and $g_1, \ldots, g_k$ by *composition*.

Composition of (Partially) Computable Functions

Theorem 1.1. If h is obtained from the (partially) computable functions $f, g_1, \ldots, g_k$ by composition, then h is (partially) computable.

Proof. The following program computes h:

$$Z_1 \leftarrow g_1(X_1, \ldots, X_n)$$
$$\ldots$$
$$Z_k \leftarrow g_k(X_1, \ldots, X_n)$$
$$Y \leftarrow f(Z_1, \ldots, Z_k)$$

If $f, g_1, \ldots, g_k$ are total, so is h. □

Recursion

Suppose k is some fixed number and

$$\begin{aligned} h(0) &= k, \\ h(t+1) &= g(t, h(t)), \end{aligned}$$

where g is some given *total* function of two variables. Then h is said to be obtained from g by *primitive recursion*, or simply *recursion*.

Recursion of Computable Functions

Theorem 2.1. If h is obtained from g as in the previous slide and let g be computable. Then then h is also computable.

Proof. The following program computes h:

$$
\begin{array}{ll}
 & Y \leftarrow k \\
[A] & \text{IF } X = 0 \text{ GOTO } E \\
 & Y \leftarrow g(Z, Y) \\
 & Z \leftarrow Z + 1 \\
 & X \leftarrow X - 1 \\
 & \text{GOTO } A
\end{array}
$$

where $Y \leftarrow k$ is expanded to k lines of $Y \leftarrow Y + 1$. □

More Recursion

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, t+1) &= g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n), \end{aligned}$$

where f is a total function of n variables, and g is a total function of $n+2$ variables. Function h of $n+1$ variable is said to be obtained from g by *primitive recursion*, or simply *recursion*, from f and g.

More Recursion of Computable Functions

Theorem 2.2. If h is obtained from g as in the previous slide and let g be computable. Then then h is also computable.

Proof. The following program computes $h(x_1, \ldots, x_n, x_{n+1})$:

$Y \leftarrow f(x_1, \ldots, x_n)$
[A] IF $X_{n+1} = 0$ GOTO E
$Y \leftarrow g(Z, Y, X_1, \ldots, X_n)$
$Z \leftarrow Z + 1$
$X_{n+1} \leftarrow X_{n+1} - 1$
GOTO A

□