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 5, 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/`

1 A Universal Program (4)

1.1 Universality (4.3)

The Stepping Function STP

A simple modification of the program $\mathscr{U}_n$ would enable us to prove that following predicate is computable:

$$\begin{aligned}\mathrm{STP}(x_1,\ldots,x_n,y,t) \;&\Leftrightarrow\; \text{Program number } y \text{ halts after}\\ &\qquad t \text{ or fewer steps on inputs } x_1,\ldots,x_n\\ &\Leftrightarrow\; \text{There is a computation of program } y\\ &\qquad \text{of length} \le t+1, \text{ beginning with}\\ &\qquad \text{inputs } x_1,\ldots,x_n\end{aligned}$$

We simply need to add a counter to determine when we have simulated t steps. However, we can prove a stronger result.

Function STP is Primitive Recursive

Theorem 3.2. For each $n > 0$, the predicate $\mathrm{STP}^{(n)}(x_1,\ldots,x_n,y,t)$ is primitive recursive. □ The idea is to provide numeric versions of the notations of snapshot and successor of snapshot, and to show that the necessary functions are primitive recursive. We first define the following functions for extracting the components of the ith instruction of program number y:

$$\begin{aligned}\mathrm{LABEL}(i,y) &= l((y+1)_i)\\ \mathrm{VAR}(i,y) &= r(r((y+1)_i))+1\\ \mathrm{INSTR}(i,y) &= l(r((y+1)_i))\\ \mathrm{LABEL}'(i,y) &= l(r((y+1)_i)) \dot{-} 2\end{aligned}$$

Function STP is Primitive Recursive, Continued

Next we define some predicates that indicate, for program y and the snapshot represented by $x = \langle i, s\rangle$, where i is the number of the instruction to be executed and s the current state (i.e., variable S in $\mathscr{U}_n$), what kind of action is to be performed next.

$$\begin{aligned}\mathrm{SKIP}(x,y) \;&\Leftrightarrow\; [\mathrm{INSTR}(l(x),y) = 0 \;\&\; l(x) \le Lt(y+1)]\\ &\qquad \vee\, [\mathrm{INSTR}(l(x),y) \ge 2 \;\&\; \sim (p_{\mathrm{VAR}_{(l(x),y)}} | r(x))]\\ \mathrm{INCR}(x,y) \;&\Leftrightarrow\; \mathrm{INSTR}(l(x),y) = 1\\ \mathrm{DECR}(x,y) \;&\Leftrightarrow\; \mathrm{INSTR}(l(x),y) = 2 \;\&\; p_{\mathrm{VAR}_{(l(x),y)}} | r(x)\\ \mathrm{BRANCH}(x,y) \;&\Leftrightarrow\; \mathrm{INSTR}(l(x),y) > 2 \;\&\; p_{\mathrm{VAR}_{(l(x),y)}} | r(x)\\ &\qquad \&\; (\exists i)_{\le Lt(y+1)} \mathrm{LABEL}(i,y) = \mathrm{LABEL}'(l(x),y)\end{aligned}$$

Function STP is Primitive Recursive, Continued

Now we can define $\mathrm{SUCC}(x,y)$, which for program number y, computes the successor to

the snapshot represented by x.

$$\mathrm{SUCC}(x,y) = \begin{cases} \langle l(x)+1,\ r(x)\rangle & \text{if } \mathrm{SKIP}(x,y) \\ \langle l(x)+1,\ r(x)\cdot p_{\mathrm{VAR}(l(x),y)}\rangle & \text{if } \mathrm{INCR}(x,y) \\ \langle l(x)+1,\ r(x)/p_{\mathrm{VAR}(l(x),y)}\rangle & \text{if } \mathrm{DECR}(x,y) \\ \langle \min_{i\le Lt(y+1)}[\mathrm{LABEL}(i,y)=\mathrm{LABEL}'(l(x),y)],\ r(x)\rangle & \text{if } \mathrm{BRANCH}(x,y) \\ \langle Lt(y+1)+1,\ r(x)\rangle & \text{otherwise.} \end{cases}$$

Function STP is Primitive Recursive, Continued

We also need

$$\mathrm{INIT}^{(n)}(x_1,\ldots,x_n) = \langle 1, \prod_{i=1}^{n}(p_{2i})^{x_i}\rangle$$

which gives the representation of the initial snapshot for input $x_1,\ldots,x_n$, and

$$\mathrm{TERM}(x,y) \Leftrightarrow l(x) > Lt(y+1)$$

which tests whether x represents a terminal snapshot for program y.

Function STP is Primitive Recursive, Continued

Putting these together we can define a primitive recursive function that gives the numbers of the successive snapshots produced by a given program:

$$\begin{aligned} \mathrm{SNAP}^{(n)}(x_1,\ldots,x_n,y,0) &= \mathrm{INIT}^{(n)}(X_1,\ldots,x_n) \\ \mathrm{SNAP}^{(n)}(x_1,\ldots,x_n,y,i+1) &= \mathrm{SUCC}(\mathrm{SNAP}^{(n)}(x_1,\ldots,x_n,y,i),y) \end{aligned}$$

Thus,

$$\mathrm{STP}^{(n)}(x_1,\ldots,x_n,y,t) \Leftrightarrow \mathrm{TERM}(\mathrm{SNAP}^{(n)}(x_1,\ldots,x_n,y,t),y)$$

It is clear that $\mathrm{STP}^{(n)}(x_1,\ldots,x_n,y,t)$ is primitive recursive. □

A Normal Form for Partial Computable Functions

Theorem 3.3. Let $f(x_1,\ldots,x_n)$ be a partially computable function. Then there is a primitive recursive predicate $R(x_1,\ldots,x_n,y)$ such that

$$f(x_1,\ldots,x_n) = l(\min_z R(x_1,\ldots,x_n,z))$$

□

Proof. Let y_0 be the number of a program that computes $f(x_1,\ldots,x_n)$. Let $R(x_1,\ldots,x_n,z)$ be the following predicate

$$\begin{aligned} & \mathrm{STP}^{(n)}(x_1,\ldots,x_n,y_0,r(z)) \\ \& \quad & (r(\mathrm{SNAP}^{(n)}(x_1,\ldots,x_n,y_0,r(z))))_1 = l(z) \end{aligned}$$

If the above predicate is defined, then for any such z, the computation by program y_0 terminates in $r(z)$ or few steps, and $l(z)$ is the value held in the output variable Y. If on the other hand, the predicate is undefined, then $\text{STP}^{(n)}(x_1, \ldots, x_n, y_0, t)$ must be false for all t. That is, $f(x_1, \ldots, x_n) \uparrow$. □

A Characterization of Partially Computable Functions

Theorem 3.4. A function is partially computable if and only if it can be obtained from the initial functions by a finite number of applications of composition, recursion, and minimalization. □ *Proof.* ($\Leftarrow$) That every function which can be so obtained is partially computable is a consequence of Theorems 1.1, 2.1, 2,2, 3.1, and 7.2 in Chapter 3. That is, there is a program in $\mathscr{S}$ for the function so obtained. ($\Rightarrow$) Given a partially computable function — a program in language $\mathscr{S}$ — Theorem 3.3 in this Chapter show how to express this function in the form

$$l(\min_z R(x_1, \ldots, x_n, z))$$

where R is a primitive recursive predicate. Finally, R is used in a minimalization and then composed with the primitive recursive function l. □

A Characterization of Computable Functions

When $\min_z R(x_1, \ldots, x_n, z)$ is a total function, we say that we are applying the operation of *proper* minimalization to R. Now, if

$$l(\min_z R(x_1, \ldots, x_n, z))$$

is total, then $R(x_1, \ldots, x_n, z)$ must be total too. Hence we have: **Theorem 3.5.** A function is computable if and only if it can be obtained from the initial functions by a finite number of applications of composition, recursion, and *proper* minimalization. □

1.2 Recursively Enumerable Sets (4.4)

Review: Sets and Characteristic Functions

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:

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

Sets and Classes of Functions

- The predicate $\mathrm{HALT}(x, y)$ is the characteristic function of the set

$$\{(x, y) \in N^2 \mid \mathrm{HALT}(x, y)\}.$$

- A set $B \subseteq N^m$ is said to belong to *some class of functions* means that the characteristic function $P(x_1, \ldots, x_n)$ of B belongs to the class in question.
- B is computable or recursive is just to say that $P(x_1, \ldots . x_n)$ is a computable function.
- B is a primitive recursive set if $P(x_1, \ldots . x_n)$ is primitive recursive.

Theorem 4.1. Let the sets B, C belong to some PRC class $\mathscr{C}$. The so do the sets $B \cup C$, $B \cap C$, and $\bar{B}$. □

Need Only Consider Subsets of N

Theorem 4.2. Let $\mathscr{C}$ be a PRC class, and let B be a subset of $N^m, m \geq 1$. Then B belongs to $\mathscr{C}$ if and only if

$$B' = \{[x_1, \ldots, x_m] \in N \mid (x_1, \ldots, x_m) \in B\}$$

belongs to $\mathscr{C}$. □. *Proof.* If $P_B(x_1, \ldots, x_m)$ is the characteristic function of B, then

$$P_{B'}(x) \Leftrightarrow P_B((x)_1, \ldots, (x)_m) \;\&\; Lt(x) \leq m \;\&\; x > 0$$

is the characteristic function of B'. Clearly, $P_{B'}$ belongs to $\mathscr{C}$ if P_B belongs to $\mathscr{C}$. On the other hand, if $P_{B'}(x)$ is the characteristic function of B', then

$$P_B(x_1, \ldots, x_m) \Leftrightarrow P_{B'}([x_1, \ldots, x_m])$$

is the characteristic function of B. Clearly, P_B belongs to $\mathscr{C}$ if $P_{B'}$ belongs to $\mathscr{C}$. □

Recursively Enumerable

Definition. The set $B \subseteq N$ is called *recursively enumerable* if there is a partially computable function $g(x)$ such that

$$B = \{x \in N \mid g(x) \downarrow\}.$$

□

- A set is recursively enumerable just when it is the domain of a partially computable function.
- If $\mathscr{P}$ is a program that computes function g above, then B is the set of all input to $\mathscr{P}$ for which $\mathscr{P}$ eventually halts.
- B can be thought of intuitively as a set for which there exists a *semi-decision procedure* to solve the membership problem of B. This algorithm answers "yes" for number $n \in B$, but never terminates for $n \notin B$.
- The term *recursively enumerable* is usually abbreviated *r.e.*

Recursive Sets

Theorem 4.3. If B is a recursive set, then B is r.e. *Proof.* Consider the following program $\mathscr{P}$

[A] IF $\sim (X \in B)$ GOTO A

Since B is recursive, the predicate $x \in B$ is computable and $\mathscr{P}$ can be expanded to a program of $\mathscr{S}$. Let $\mathscr{P}$ computes the function $h(x)$. Then, clearly,

$$B = \{x \in N \mid h(x) \downarrow\}.$$

□

What If Both B and $\bar{B}$ Are r.e.?

Theorem 4.4. The set B is recursive if and only if B and $\bar{B}$ are both r.e. *Proof.* ($\Rightarrow$) If B is recursive, then by Theorem 4.1 so is $\bar{B}$. By Theorem 4.3, they are both r.e. ($\Leftarrow$) If both B and $\bar{B}$ are r.e., then there are programs $\mathscr{P}$ and $\mathscr{Q}$ such that

$$\begin{aligned} B &= \{x \in N \mid \Psi^{(1)}_{\mathscr{P}}(x) \downarrow\} \\ \bar{B} &= \{x \in N \mid \Psi^{(1)}_{\mathscr{Q}}(x) \downarrow\} \end{aligned}$$

Then B is recursive as it is computed by the following program:

[A] IF $\text{STP}^{(1)}(X, \#(\mathscr{P}), T)$ GOTO C
IF $\text{STP}^{(1)}(X, \#(\mathscr{Q}), T)$ GOTO E
$T \leftarrow T + 1$
GOTO A
[C] $Y \leftarrow 1$

□

The Intersection of Two r.e. Sets

Theorem 4.5. If B and C are r.e. sets so are $B \cap C$ and $B \cup C$. *Proof.* Let

$$\begin{aligned} B &= \{x \in N \mid g(x)\downarrow\} \\ C &= \{x \in N \mid h(x)\downarrow\} \end{aligned}$$

where g and h are partially computable. Let $f(x)$ be the function computed by the program

$Y \leftarrow g(X)$
$Y \leftarrow h(X)$

Hence

$$B \cap C = \{x \in N \mid f(x)\downarrow\}$$

hence $B \cap C$ is r.e.

The Union of Two r.e. Sets

Proof. (Continued) Let g and h be computed by programs $\mathscr{P}$ and $\mathscr{Q}$, respectively. Let $k(x)$ be the function computed by the program:

$[A]$ IF $\text{STP}^{(1)}(X, \#(\mathscr{P}), T)$ GOTO E
IF $\text{STP}^{(1)}(X, \#(\mathscr{Q}), T)$ GOTO E
$T \leftarrow T + 1$
GOTO A

Then $k(x)$ is defined just in case *either* $g(x)$ *or* $h(x)$ is defined. That is,

$$B \cup C = \{x \in N \mid k(x)\downarrow\}$$

so that $B \cup C$ is also r.e. □