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 7, 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/`

1 A Universal Program (4)

1.1 Diagonalization and Reducibility (4.5)

Diagonalization and Reducibility

- Diagonalization and reducibility are two general techniques for proving that given sets are not recursive or even that they are not r.e.
- Diagonalization shows an object $b \notin A$ by

1. first demonstrating that the set A can be enumerated in a suitable way,
2. then, with the help of the enumeration, defining an object b that is different from every object in the enumeration of A.

- Reducibility transforms the membership problem of a set A to the membership problem of another set B, hence showing that testing membership in A is "no harder than" testing membership in B.

Diagonalization

- Diagonalization shows an object $b \notin A$ by
  1. first demonstrating that the set A can be enumerated in a suitable way,
  2. then, with the help of the enumeration, defining an object b that is different from every object in the enumeration of A.
- We says that b is defined by *diagonalizing over* A.
- Often there is an additional twist: The definition of b is such that b *must belong to* A, contradicting the assertion that we began with an enumeration of *all* elements in A.
- We then draw some conclusion from this contradiction.

Diagonalization, Example 1

The predicate HALT(x, y) is not computable. *Proof.* Assume the predicate HALT(x, y) is computable. Then we can write a program $\mathscr{P}$ in language $\mathscr{S}$ as follows:

$[A]$ IF HALT(X, X) GOTO A

We now show by diagonalization the following contradiction.

1. There is an enumeration of all the programs expressible in $\mathscr{S}$: $\mathscr{P}_0, \mathscr{P}_1, \ldots,$ ...
2. Function $\Psi^{(1)}_{\mathscr{P}}$ differs from each function $\Psi^{(1)}_{\mathscr{P}_0}, \Psi^{(1)}_{\mathscr{P}_1}, \ldots$ on at least one input value: For each program $\mathscr{P}_n, n \in N$,

   $\Psi^{(1)}_{\mathscr{P}_n}(n)\uparrow$ if and only if $\Psi^{(1)}_{\mathscr{P}}(n)\downarrow$

There shows that $\mathscr{P}$ is not in the enumeration, which is a contradiction. We conclude that HALT(x, y) is not computable. □

Diagonalization, Example 1 Continued

Given the following program $\mathscr{P}$:

$[A]$ IF HALT(X, X) GOTO A

Function $\Psi^{(1)}_{\mathscr{P}}$ differs from each function $\Psi^{(1)}_{\mathscr{P}_0}, \Psi^{(1)}_{\mathscr{P}_1}, \ldots$ along the diagonal of the following array representation of all the functions expressible by programs in language $\mathscr{S}$:

$$\begin{array}{cccc}
\boxed{\Psi^{(1)}_{\mathscr{P}_0}(0)} & \Psi^{(1)}_{\mathscr{P}_0}(1) & \Psi^{(1)}_{\mathscr{P}_0}(2) & \ldots \\
\Psi^{(1)}_{\mathscr{P}_1}(0) & \boxed{\Psi^{(1)}_{\mathscr{P}_1}(1)} & \Psi^{(1)}_{\mathscr{P}_1}(2) & \ldots \\
\Psi^{(1)}_{\mathscr{P}_2}(0) & \Psi^{(1)}_{\mathscr{P}_2}(1) & \boxed{\Psi^{(1)}_{\mathscr{P}_2}(2)} & \ldots \\
\vdots & \vdots & \vdots &
\end{array}$$

Diagonalization, Example 2

Let TOT be the set of all numbers p such that p is the number of a program that computes a total function $f(x)$ of one variable. That is,

$$\text{TOT} = \{z \in N \mid (\forall x)\Phi(x, z) \downarrow\}$$

Theorem 6.1. TOT is not r.e. *Proof.* Assume TOT is r.e. By Theorem 4.9, there is a computable function $g(x)$ such that TOT $= \{g(0), g(1), g(2), \ldots\}$. Let

$$h(x) = \Phi(x, g(x)) + 1$$

As for each x, $\Phi(x, g(x)) \downarrow$, function h is itself computable. Let h be computed by a program $\mathscr{P}$, and let $p = \#(\mathscr{P})$. Then $p \in$ TOT, so that $p = g(i)$ for some i. However,

$$h(i) = \Phi(i, g(i)) + 1 = \Phi(i, p) + 1 = h(i) + 1$$

which is a contradiction. We conclude that TOT is not r.e. □

Many-one Reducibility

Definition. Let A, B be sets. A is *many-one reducible to* B, written $A \leq_m B$, if there is a computable function f such that

$$A = \{x \in N \mid f(x) \in B\}$$

That is, $x \in A$ if and only if $f(x) \in B$. Note that f need not be one-one. If $A \leq_m B$, then in a sense testing membership in A is "no harder than" testing membership in B. In particular, to test $x \in A$, we can compute $f(x)$ and then test $f(x) \in B$.

Main Theorem of Reducibility

Theorem 6.2. Suppose $A \leq_m B$.

1. If B is recursive, the A is recursive.

2. If B is r.e., then A is r.e.

Proof.

1. Let $A = \{x \in N \mid f(x) \in B\}$, where f is computable, and let $P_B(x)$ be the characteristic function over B. Then

   $$A = \{x \in N \mid P_B(f(x))\},$$

   Since $P_B(x)$ is recursive, the characteristic function of A, $P_B(f(x))$, is also recursive.

2. Now suppose that B is r.e.. Then $B = \{x \in N \mid g(x) \downarrow\}$ for some partially computable function g, and $A = \{x \in N \mid g(f(x)) \downarrow\}$. But $g(f(x))$ is partially computable, so A is r.e.

□

Applying Reducibility

If $A \leq_m B$, then

1. If A is not recursive, the B is not recursive.

2. If A is not r.e., then B is not r.e.

That is, to show that a set B is not recursive (r.e.), we find a set A that is not recursive (r.e.) and proceed to show that $A \leq_m B$.

Applying Reducibility, Example

In order to show that K_0, defined by

$$K_0 = \{x \in N \mid \Phi_{r(x)}(l(x)) \downarrow\} = \{\langle x, y\rangle \mid \Phi_y(x) \downarrow\},$$

is not recursive. We need only to show that $K \leq_m K_0$, where

$$K = \{n \in N \mid n \in W_n\}.$$

is known to be not recursive. Let function $f(x) = \langle x, x\rangle$. Clearly $f(x)$ is computable. Then $K \leq_m K_0$ because $x \in K$ if and only if $\langle x, x\rangle \in K_0$. As K is not recursive, neither is K_0.

m-completeness

Definition A set A is *m-completeness* if

1. A is r.e., and

2. for every r.e. set B, $B \leq_m A$.

Example: K_0 is m-complete. That is because if a set B is r.e., then

$$\begin{aligned} B &= \{x \in N \mid g(x)\downarrow\} && \text{for some partially computable } g \\ &= \{x \in N \mid \Phi(x, z_o)\downarrow\} && \text{for some } z_0 \\ &= \{x \in N \mid \langle x, z_0\rangle \in K_0\} \end{aligned}$$

That is, $B \leq_m K_0$, so by definition K_0 is m-complete.

More Theorems of Reducibility

Theorem 6.3. If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$. *Proof.* Let $A = \{x \in N \mid f(x) \in B\}$ and $B = \{x \in N \mid g(x) \in C\}$. Then $A = \{x \in N \mid g(f(x)) \in C\}$, and $g(f(x))$ is computable. □ **Corollary 6.4.** If A is m-complete, B is r.e., and $A \leq_m B$, then B is m-complete. *Proof.* If C is r.e. then by assumption $C \leq_m A$, and $A \leq_m B$. It follows that $C \leq_m B$, hence B is m-complete. □ **Definition** $A \equiv_m B$ means that $A \leq_m$ and $B \leq_m A$.

$K_0 \leq_m K$

To prove $K_0 \leq_m K$, we need to find a computable function f such that $f(\langle n, q\rangle)$ is the number of a program with the following property

$$\Phi_q(n)\downarrow \quad \Leftrightarrow \quad \Phi_{f(\langle x,q\rangle)}(f(\langle x, q\rangle))\downarrow$$

Let $\mathscr{P}$ be the program

$$Y \leftarrow \Phi^{(1)}(l(X_2), r(X_2))$$

and let $p = \#(\mathscr{P})$. Therefore,

$$\begin{aligned} \Phi_q(n) &= \Psi_{\mathscr{P}}(x_1, \langle n, q\rangle) = \Phi^{(2)}(x_1, \langle n, q\rangle, p) \\ &= \Phi^{(1)}(x_1, S_1^1(\langle n, q\rangle, p)) = \Phi_{S_1^1(\langle n,q\rangle,p)}(x_1) \end{aligned}$$

for all x_1. In particular, when $x_1 = S_1^1(\langle n, q\rangle, p)$ we arrive at

$$\Phi_q(n) = \Phi_{S_1^1(\langle n,q\rangle,p)}(S_1^1(\langle n, q\rangle, p))$$

That is, $\langle n, q\rangle \in K_0$ if and only if $S_1^1(\langle n, q\rangle, p) \in K$. As $f(\langle n, q\rangle) = S_1^1(\langle n, q\rangle, p)$ is computable, we conclude $K_0 \leq_m K$.

$K_0 \equiv_m K$

Theorem 6.5.

1. K and K_0 are m-complete.

2. $K \equiv_m K_0$.

Reducibility, More Example

Theorem 6.6. The set EMPTY $= \{x \in N \mid W_x = \emptyset\}$ is not r.e. *Proof.* We will show $\bar{K} \leq_m$ EMPTY. As $\bar{K}$ is not r.e., so neither is EMPTY. Let $\mathscr{P}$ be the program $\boxed{Y \leftarrow \Phi(X_2, X_2)}$ and let $p = \#(\mathscr{P})$. As $\mathscr{P}$ ignores its first argument, so for a given z,

$$(\forall x)(\Psi_{\mathscr{P}}^{(2)}(x, z)\downarrow) \quad \text{if and only if} \quad \Phi(z, z)\downarrow$$

By the parameter theorem

$$\Psi_{\mathscr{P}}^{(2)}(x_1, x_2) = \Phi^{(2)}(x_1, x_2, p) = \Phi^{(1)}(x_1, S_1^1(x_2, p))$$

Therefore, for any z,

$$\begin{aligned} z \in \bar{K} \quad & \text{if and only if} \quad \Phi(z, z)\uparrow \\ & \text{if and only if} \quad \Phi^{(1)}(x, S_1^1(z, p))\uparrow \text{ for all } x \\ & \text{if and only if} \quad W_{S_1^1(z,p)} = \emptyset \\ & \text{if and only if} \quad S_1^1(z, p) \in \text{EMPTY} \end{aligned}$$

As $f(z) = S_1^1(z, p)$ is computable, so we have $\bar{K} \leq_m$ EMPTY.

1.2 Rice's Theorem

Are There Many Not Recursive Sets?

Let Γ be some collection of partially computable functions of one variable. We may associate with Γ the set (usually called an *index set*)

$$R_\Gamma = \{t \in N \mid \Phi_t \in \Gamma\}.$$

R is a recursive set just in case the predicate $g(t)$, defined as $g(t) \Leftrightarrow \Phi_t \in \Gamma$, is computable. Invoking Church's thesis, we can say that R_Γ is a recursive set just in case there is an algorithm that accepts *programs* $\mathscr{P}$ as input and returns the value TRUE or FALSE depending on whether or not the function $\Psi_{\mathscr{P}}^{(1)}$ does or does not belong to Γ. We will show that R_Γ is almost always not recursive.

Rice's Theorem

Theorem 7.1. let Γ be a collection of partially computable functions of one variable. Let there be partially computable functions $f(x)$ and $g(x)$ such that $f(x)$ belongs to Γ but $g(x)$ does not. Then R_Γ is not recursive. *Proof.* Let $h(x)$ be the function such that $h(x)\uparrow$ for all x. We assume first that $h(x)$ does not belong to Γ. Let q be the number of

```
$Z \leftarrow \Phi(X_2, X_2)$
$Y \leftarrow f(X_1)$
```

Then, for any i, $S_1^1(i,q)$ is the number of

```
$X_2 \leftarrow i$
$Z \leftarrow \Phi(X_2, X_2)$
$Y \leftarrow f(X_1)$
```

Rice's Theorem, Proof Continued

Proof. (Continued) Now

$$\begin{aligned} i \in K \quad \Rightarrow \quad \Phi(i,i)\downarrow \ &\Rightarrow\ \Phi_{S_1^1(i,q)}(x) = f(x) \ \text{ for all } x \\ &\Rightarrow\ \Phi_{S_1^1(i,q)}(x) \in \Gamma \\ &\Rightarrow\ S_1^1(i,q) \in R_\Gamma, \end{aligned}$$

and

$$\begin{aligned} i \notin K \quad \Rightarrow \quad \Phi(i,i)\uparrow \ &\Rightarrow\ \Phi_{S_1^1(i,q)}(x)\uparrow \ \text{ for all } x \\ &\Rightarrow\ \Phi_{S_1^1(i,q)}(x) = h \notin \Gamma \\ &\Rightarrow\ S_1^1(i,q) \notin R_\Gamma, \end{aligned}$$

so $K \leq_m R_\Gamma$. By theorem 6.2, R_Γ is not recursive. If $h(x)$ does belongs to Γ, then the same argument with Γ and $f(x)$ replaced by $\bar{\Gamma}$ and $g(x)$ shows that $R_{\bar{\Gamma}}$ is not recursive. But $R_{\bar{\Gamma}} = \bar{R}_\Gamma$, so, by Theorem 4.1, R_Γ is not recursive in this case either. □

Rice's Theorem, Examples

Consider the following collections of partially computable functions:

1. Γ is the set of computable functions;
2. Γ is the set of primitive recursive functions;
3. Γ is the set of partially computable functions that are defined for all but a finite number of values of x.

Is the set R_Γ recursive in each of the above three cases? None of them is recursive. To see why, for example, simply let $f(x) = u_1^1(x)$ and $g(x) = 1 - x$ and invoke the Rice's theorem. (Note that $g(x)$ is defined only for $x = 0, 1$.)

Rice's Theorem, Implications

We often wish to develop algorithms — that is, programs — that will accept a program as input and will return as output some useful property of the partial function computed by that program. Of course, for the algorithms (programs) to be useful, they must terminate for all input. However, when this property is sufficiently interesting — that is, some function has this property but some has not — then by Rice's Theorem, there exists no such algorithm (program) to tell whether the input program has such property or not.