Lecture Notes Recursion Theory

Form

Homework once or twice a week. The deadline is generally almost a week later: either Monday 15 hours sharp, or Thursday 9. You are not supposed to miss anything because you are still writing up homework. The best 80% (thereabouts) of the grades count. Cooperation is *absolutely not* forbidden; however, we prefer to have things written up in your own words. Written exam in December. Final grade is the average of homework grade and exam, as long as the exam grade is at least 5. *Special arrangements will be made whenever this seems reasonable.*

1 Algorithms

Examples. Adding a list of integers in decimal notation (Renaissance): 1234 + 5678 + 9012 = …. Euclid's Algorithm for finding the g.c.d (± 300 b.C.): (1234, 5678) = 2. Sieve of Eratosthenes, finding the prime numbers below a given number n (± 240 b.C.).

An algorithm is a procedure specified by a finite set of instructions that can be applied in a deterministic fashion. We have an algorithm for function f if for any $x \in \mathrm{Dom} f$, we end up, starting from x, with $f(x)$ after a finite number of steps.

Before 1930, no need was felt for a *definition* of the algorithmically (or *effectively*) calculable functions. Rather, it as assumed that clearly stated problems, such as whether a Diophantine equation

$$P(x_0,\ldots, x_{n-1}) = 0,$$

where P is a polynomial with integer coefficients, or whether a given formula of first order logic is universally valid, would have general solutions that one would recognize as algorithmic.

In 1931, Gödel proved that any effectively axiomatized formal system T that includes elementary number theory is either inconsistent, or incomplete (first incompleteness theorem). In fact, one can code formulas into numbers, and thus find a formula φ which says 'T is consistent'; and this φ cannot be proved in T — unless it is false (second incompleteness theorem). A fortiori there is no effective system that can decide whether a first order statement on number theory is true or not.

This signal failure of a research project of Hilbert's suggested to some mathematicians that other problems might fail to have algorithmic solutions as well. But to *prove* such things, one would need a definition.

2 Partial functions

We could describe algorithms for functions $\omega \to \omega$ in some formalism, e.g. the English language; by ordering them alphabetically we would effectively get a list $(f_n)_n$ of unary functions. Now we could define a function g by: $g(n) = f_n(n) + 1$. How can we have forgotten g? — This is a variant of *Richard's Paradox*.

We insist that we will succeed in inventing an unambiguous formalism in which we can effectively describe all algorithms. Then the above argument shows *that we will not be able to see if the function defined by a given algorithm is total*. We must admit *partial functions*.

3 λ-definability

Assume we have certain things that we can apply to each other, as a function to an argument; write $x \cdot y$ for 'the result of applying x to y'. If $M = M(x)$ is an expression, the meaning of which may be supposed to depend on the denotation of the variable x, write $\lambda x.M$ to denote the function that, applied to some thing a, returns $M(a)$. Church invented this notation, with some rules for its use, around 1930. In 1931, few functions had been shown to be definable in it.

4 Primitive recursion

One can define functions by *recursion*: making use of their values for smaller arguments. For example, the inductive definition of addition runs

$$0 + x = x,$$
$$(y + 1) + x = (y + x) + 1;$$

this, in view of the fact that 0 and adding 1 will get you every natural number, says it all.

Definition. The *primitive recursive functions* are the elements of the least class C of functions containing
(I) the *successor function* $\lambda x.\ x + 1$;
(II) the *constant function* $\lambda x.\ 0$;
(III) the *projection functions* e^n_i, for $n \geq 1$ and $1 \leq i \leq n$, defined by

$$e^n_i(x_1,\ldots, x_n) = x_i;$$

and closed under the schemes of
(IV) *composition*, which takes functions $f, g_1,\ldots, g_n$, where n is the arity of f and $g_1,\ldots, g_n$ have the same arity, and produces the function $h = f(g_1,\ldots, g_n)$ defined by

$$h(\overline{x}) = f(g_1(\overline{x}),\ldots, g_n(\overline{x}));$$

(V) *primitive recursion*, which takes functions g and h, where the arity of h is 2 + the arity of g, and produces the function $f = \mathsf{P}(g, h)$ defined by

$$f(0, \bar{x}) = g(\bar{x}),$$
$$f(y + 1, \bar{x}) = h(y, f(y, \bar{x}), \bar{x}).$$

Examples. (i) Instead of $f(g)$ we may write $f \circ g$; we abbreviate $\lambda x.\ x + 1$ to S. Then $\lambda(x_1,\ldots, x_n).m = S \circ \ldots \circ S \circ (\lambda x.0) \circ e^n_1$. We can *prove* $\lambda(x_1,\ldots, x_n).m$ primitive recursive by induction on m, as follows:
($m = 0$) e^n_1 is prim. by (III); hence $(\lambda x.0) \circ e^n_1$ is prim. by (IV); and this is the intended function, for $((\lambda x.0) \circ e^n_1)(x_1,\ldots, x_n) = (\lambda x.0)(x_1) = 0$.
($m = k + 1$) If $f = \lambda(x_1,\ldots, x_n).k$, then $\lambda(x_1,\ldots, x_n).m = S \circ f$.
(ii) $\lambda(x, y).\ x + y = \mathsf{P}(e^1_1, S \circ e^3_2)$. Derivation: e^3_2 is prim. by (III); hence $f = S \circ e^n_1$ is prim. by (IV); e^1_1 is prim. by (III); so $\mathsf{P}(e^1_1, f)$ is prim. by (V).

Likewise, $x \cdot y$ (multiplication), x^y (exponentiation), $x!$ (x *factorial*, $x! = x \cdot (x - 1) \cdot \ldots \cdot 1, 0! = 1$) , $x \dot{-} y$ (*monus*, $x - y$ if $x > y$ and 0 otherwise).

A predicate (relation) R is primitive recursive if its characteristic function χ_R is.

Example. The set of primes is primitive recursive.

Let $p_0, p_1,\ldots$ be the primes in increasing order. Every positive integer x has a unique representation

(*) $$x = p_0^{n_0} \cdot \ldots \cdot p_k^{n_k} \cdot \ldots$$

The function $(x)_k = n_k$ (the exponent of p_k in (*)) is primitive recursive. This sort of thing will enable us to code things as natural numbers.

5 Exercises

:1 Check the examples after 'Likewise' above. Conclude that the primitive recursive relations are closed under conjunction.

:2. Use monus to show that $\overline{sg}(x) := 0 \lhd (x > 0) \rhd 1$ is primitive recursive. Conclude that the primitive recursive relations are closed under negation, and hence under all the Boolean operations.

:3 (Definition by cases). If $g_1,\ldots, g_n$ are primitive recursive functions, and $R_1,\ldots, R_n$ mutually exclusive primitive recursive relations, all of the same arity, then f, defined by

$f(\bar{x}) = g_1(\bar{x})$ if $R_1(\bar{x})$,
…
$f(\bar{x}) = g_n(\bar{x})$ if $R_n(\bar{x})$,

is primitive recursive.

:4 If f is primitive recursive, then so are $\prod_{y<x} f(y, \bar{x})$ and $\sum_{y<x} f(y, \bar{x})$.

:5 If R is primitive recursive, then so are $\forall y{<}z\ R(y, \bar{x})$ and $\exists y{<}z\ R(y, \bar{x})$. If R is primitive recursive, then so is $\mu y{<}z\ R(y, \bar{x})$.

:6 Conclude from :5 that the set of primes and the function $(x)_k$ are primitive recursive.

Literature

S. Barry Cooper: *Computability Theory*. Londen, 2004.

Martin Davis: *Computability and unsolvability*. 1958/ New York 1982. With appendices: Results from the theory of numbers, & Hilbert's tenth problem is unsolvable.

S.C. Kleene: *Introduction to metamathematics*. Amsterdam, 1952.

Hartley Rogers Jr.: *Theory of recursive functions and effective computability*. New York 1967.

Robert Soare: *Computability Theory and Applications*, to be distributed.

The original version of this book was published as *Recursively enumerable sets and degrees* by Springer in 1987.