Logic, Computability and Incompleteness.

Exercise Set 1. Solutions

1) Show that the set of odd positive integers is enumerable, and provide an enumerating function $f(x)$.

To establish enumerability, need to show that the elements of the set of odd positive integers can all be arranged in a single list, i.e. the set constitutes the range of an onto function from the positive integers. Here's a pictorial representation of such a list:

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5\ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 3 & 5 & 7 & 9\ldots \end{array}$$

where $f(x) = 2x - 1$ is the enumerating function.

2) Show that the cardinal number of the set of squares of natural numbers is $\aleph_0$.

We can take it as read that the cardinality of the natural numbers is $\aleph_0$. To establish that the cardinality of the set of squares of natural numbers is also $\aleph_0$, need to show that there is a bijection between the two sets. The function $f(x) = x^2$ from natural numbers to their squares is both 1-1 and onto, and hence serves the purpose.

3) Show that the set of all finite strings of 0's and 1's is enumerable.

Need to show that all such finite strings can be arranged in a single list.
Here's a list: 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101,110, 111, 0000, 0001,...
Description of list: First list strings of length 1, length 2, etc. Within each length group, arrange strings in lexicographic order, where 0 comes before 1.

4) Show that the set of all infinite strings of 0's and 1's is <u>not</u> enumerable.

<u>Reductio proof</u>: Take any purportedly complete list of infinite strings, say L (where the place in the list is given by the vertical axis and number in red, while the nth element in a particular string is labelled by the numbers in green along the horizontal axis)

```
   123456789 …______....
1. 01101011101010001....
2. 00101101111010010....
3. 10110100110100000....
4. 11101011010111010....
⋮
```

and construct the <u>antidiagonal</u> string S_D such that for each string S_n on the list, the nth element of S_D is 0 if the nth element of S_n is 1, and it is 1 if the nth element of S_n is 0. S_D is a well defined infinite string of 0's and 1's, but it cannot appear anywhere on L. For suppose S_D were the mth string S_m for some arbitrary m. Then consider the mth element of S_m. If it is 1, then by construction the mth element of S_D is 0 and $S_m \neq S_D$. And if the mth element of S_m is 0 then by construction the mth element of S_D is 1 and again $S_m \neq S_D$. Hence S_D can appear nowhere on the list, contradicting the supposition that L is complete. Since the choice of L was arbitrary there can be no complete list □

5) Suppose that the set Γ is denumerable, that the set $\Delta \subset \Gamma$ is finite, and that Γ^0 is the result of removing all the elements of Δ from Γ. Prove that Γ^0 is still denumerable.

Reductio proof: As described $\Gamma = \Delta \cup \Gamma^0$, where Δ is finite. If Γ^0 were not denumerable then it would have to be finite as well, since $|\Gamma^0| \leq |\Gamma|$. But the union of two finite sets is always finite (easy proof), contradicting the supposition that Γ is denumerable.

Direct proof: Since Γ is denumerable there must exist a bijection $f(x)$ from the positive integers to Γ. Since Δ is finite its cardinality is given by some natural number n. Let Δ' be the set consisting of the first n elements of Γ under $f(x)$, and let $\Gamma' = \Gamma - \Delta'$. $|\Delta'| = |\Delta| = n$, and so $|\Gamma'| = |\Gamma^0|$. The set of positive integers $> n$ is obviously denumerable, and a bijection between all positive integers and this proper subset is given by the function $g(x) = x + n$. In turn, a bijection between the positive integers and Γ' is given by the function $f(g(x))$. So Γ' is denumerable, and since $|\Gamma'| = |\Gamma^0|$ so is Γ^0.

6) What if *denumerably* many elements are removed from Γ?

The cardinality of the resulting set will depend on *which* denumerable subset is removed. $\Gamma \subseteq \Gamma$, and if Γ is subtracted from itself, the result is the empty set with cardinality 0. Take some element $e \in \Gamma$ and let $\Delta = \Gamma - \{e\}$. Then Δ is denumerable, and $\Gamma - \Delta = \{e\}$, which has cardinality 1. Similarly take two elements $e_1, e_2 \in \Gamma$ and let $\Delta' = \Gamma - \{e_1, e_2\}$. Then Δ' is denumerable, and $\Gamma - \Delta' = \{e_1, e_2\}$, which has cardinality 2. So the result could have any finite cardinality. And since Γ is denumerable, there must exist a bijection between Γ and a proper subset of itself. So consider the function $f(x)$ from the positive integers to Γ. Let $g(x) = 2x$ and Δ be defined as the set of all $e \in \Gamma$ such that $f(g(x)) = e$. Δ is denumerable, and (by the obvious odd/even correspondence) $\Gamma - \Delta$ is denumerable as well.

7) Using quadruple notation and the conventions introduced in the Lecture Slides (topic 2, slide 16), construct a Turing Machine that computes the 1-place function f of positive integers, such that $f(x) = x + 1$. Exhibit the sequence of configurations for computing the value $f(4)$.

Here's one such machine: $q_1\ 1\ L\ q_1$
$q_1\ B\ 1\ q_2$

And here's the sequence of configurations for computing $f(4) = 5$:

start $\ldots 0\underline{1}1110\ldots$
1
$\ldots\underline{0}11110\ldots$
1
$\ldots\underline{1}11110\ldots$
2 halt

8) Using the same conventions as above, construct a single Turing Machine that computes the family of functions g^n of positive integers, such that for all n-tuples $x_1, ..., x_n$, $g^n(x_1, ..., x_n) = 1$. Exhibit the sequence of configurations for computing the value $g^2(2,3)$.

The idea is to design a machine that moves right and erases the inputs (of whatever arity) until it encounters two consecutive 0's, then halts reading a 1. Here's one such machine:

q_1 1 B q_2
q_2 B R q_1
q_1 B R q_3
q_3 1 B q_2
q_3 B 1 q_4

And here's the sequence of configurations for computing $g^2(2,3) = 1$:

start ...0$\underline{1}$1011100...
1
...0$\underline{0}$1011100...
2
...00$\underline{1}$011100...
1
...00$\underline{0}$011100...
2
...000$\underline{0}$11100...
1
...0000$\underline{1}$1100...
3
...0000$\underline{0}$1100...
2
...00000$\underline{1}$100...
1
...00000$\underline{0}$100...
2
...000000$\underline{1}$00...
1
...000000$\underline{0}$00...
2
...0000000$\underline{0}$0...
1
...00000000$\underline{0}$...
3
...00000000$\underline{1}$...
4 halt

9) Show that the set of all Turing Machines with a read/write vocabulary of symbols from the finite list $S_0, S_1, S_2, \ldots, S_n$ is enumerable.

There are different ways to do this, but the basic principle is that a TM is a finite string of symbols from an enumerable alphabet R, L, S_0, q_1, S_1, q_2, S_2, q_3, S_3,…, and all such finite strings are enumerable. See first solution to exercise 2-4 (d) in B&J (p. 18), as well as their second strategy, also used in chapter 5 (p. 46) and in the Lecture Slides on Uncomputability.

10) The *factorial* of y is the product of all the positive integers up to and including y (but is taken to be 1 when y is 0). Assuming **prod** as a primitive recursive function, provide a formal specification of the factorial function **fac**(y) in primitive recursive terms.

Informally, y *factorial* (written $y!$) is given by the recursive equations: $0! = 1$
$$y'! = y' \cdot y!$$

The schema for a 1-place **p.r**. function $h(y)$ is
$$h(0) = \mathbf{c}$$
$$h(\mathbf{s}(y)) = g(y, h(y))$$

So to express **fac** in primitive recursive format, where **fac** = **Pr[** **c**, g **]**

first let

fac(0) = 1 = s(0)

next, note that **fac**(**s**(y)) = s(y) · **fac**(y) = **prod**(s(y), **fac**(y)) so that

g(y, **fac**(y)) = **Cn[prod**, **Cn**[s,$\mathbf{id}^2_1$],$\mathbf{id}^2_2$] (y, **fac**(y))

This yields the official **p.r.** function **Pr**[s(0), **Cn**[**prod**, **Cn**[s,$\mathbf{id}^2_1$],$\mathbf{id}^2_2$]]

11) Consider the 3-place function on natural numbers $h(x_1, x_2, y) = (x_1 \cdot x_2) + y$. The function h could be expressed purely in terms of composition. Instead, formally define $h(x_1, x_2, y)$ using the primitive recursion schema [assuming **prod** and other primitive recursive functions formally defined in the lecture slides].

.

The schema for a 3-place **p.r**. function $h(x_1, x_2, y)$ is
$$h(x_1, x_2, 0) = f(x_1, x_2)$$
$$h(x_1, x_2, \mathbf{s}(y)) = g(x_1, x_2, y, h(x_1, x_2, y))$$

So to express the above function in primitive recursive format, where h = **Pr[** f, g **]**

first let

$h(x_1, x_2, 0) = f(x_1, x_2) = \mathbf{prod}(x_1, x_2)$

next, note that

$h(x_1, x_2, \mathbf{s}(y)) = \mathbf{s}(h(x_1, x_2, y))$

so let

$h(x_1, x_2, \mathbf{s}(y)) = g(x_1, x_2, y, h(x_1, x_2, y)) =$ **Cn**[s, $\mathbf{id}^4_4$] $(x_1, x_2, y, h(x_1, x_2, y))$

This yields the official **p.r.** function **Pr**[**prod**, **Cn**[s, $\mathbf{id}^4_4$]]

12) Consider the function $f(x,y)$ given in terms of the following definition by cases:

$f(x,y) = \{x \dot{-} y$ if $x > y$
$= \{x + y$ if $x < y$
$= \{$**fac**(x) if $x = y$

Provide a formal specification of f as a primitive recursive function [assuming **dif**, **sum** and other primitive recursive functions already formally defined].

.

To put $f(x,y)$ in **p.r.** terms, need a **p.r.** characteristic function for each of the 3 conditions. For the condition $x > y$ can use **sg**($x \dot{-} y$), for $x < y$ can use **sg**($y \dot{-} x$), and for $x = y$ can use <u>**sg**</u>($x \dot{-} y$) · <u>**sg**</u>($y \dot{-} x$)

Thus

$f\,(x, y) =$

$$[(x \dot{-} y) \cdot \mathbf{sg}(x \dot{-} y)] + [(x + y) \cdot \mathbf{sg}(y \dot{-} x)] + [\mathbf{fac}(x) \cdot (\underline{\mathbf{sg}}(x \dot{-} y) \cdot \underline{\mathbf{sg}}(y \dot{-} x))]$$

This would be a natural and elegant place to stop, but the next stage in the formal decomposition is $f\,(x, y) =$

$$\mathbf{prod}[\mathbf{dif}(x,y),\mathbf{sg}(\mathbf{dif}(x,y)] + \mathbf{prod}[\mathbf{sum}(x,y),\mathbf{sg}(\mathbf{dif}(y,x))] + \\ \mathbf{prod}[\mathbf{fac}(x),\mathbf{prod}(\underline{\mathbf{sg}}(\mathbf{dif}(x,y),\underline{\mathbf{sg}}(\mathbf{dif}(y,x)))]$$

Next it is possible to replace '+' with our already defined **p**.**r**. function **sum** (associating from the left) to yield: $f\,(x, y) =$

$$\mathbf{sum}[\mathbf{sum}[\mathbf{prod}[\mathbf{dif}(x,y),\mathbf{sg}(\mathbf{dif}(x,y)], \mathbf{prod}[\mathbf{sum}(x,y),\mathbf{sg}(\mathbf{dif}(y,x))]] , \\ \mathbf{prod}[\mathbf{fac}(x),\mathbf{prod}(\underline{\mathbf{sg}}(\mathbf{dif}(x,y),\underline{\mathbf{sg}}(\mathbf{dif}(y,x)))]]$$

Last, it would be possible to finish the reduction by using a number of explicit applications of composition. This final stage is left as an exercise for those who are so inclined.