

PHIL 3something - Logic II

Scott Saunders - 10163541

Winter 2016

Misc. Notation

- The set of positive integers $\{x : x \text{ is a positive integer} \}$
- The set of positive integers less than 3 $\{x : x \text{ is a positive integer and } x \text{ is less than } 3 \} = \{1, 2\}$.
- The empty set: \emptyset or Δ
- Member of: $A \subseteq B$ iff $\forall x(x \in A \implies x \in B)$
- Union of A and B: $A \cup B$ iff $\{x : x \in A \vee x \in B\}$
- Intersection of A and B: $A \cap B$ iff $\{x : x \in A \wedge x \in B\}$
- Difference of A and B: $\{x : x \in A \wedge x \notin B\}$
- For any non-empty sets A, B: Cartesian product: A of B: $A \times B: \{ \langle x, y \rangle : x \in A \wedge y \in B \}$ (ALL OF THE POSSIBILITIES)
- TOTAL FUNCTION: Every element in the domain is valid
- PARTIAL FUNCTION: Not every element in the domain is valid.
- for any set of sets A:
 - $\cup A = \{x : \exists y(y \in A \wedge x \in y)\}$
 - $\cap A = \{x : \forall y(y \in A \rightarrow x \in y)\}$
- Relations: R is
 - reflexive : $\forall x Rxx$
 - symmetric : $\forall x \forall y (Rxy \implies Ryx)$
 - transitive : $\forall x \forall y \forall z ((Rxy \wedge Ryz) \implies Rxz)$
 - Euclidean : $\forall x \forall y \forall z ((Rxy \wedge Rxz) \implies Ryz)$
 - a equivalence relation : it's symmetric, reflexive, transitive.
 - a (partial) function : $\exists x$ and there is at most one y: Rxy : denoted f
 - a (partial) function¹ : $\exists x, \exists y | Rxy$: denoted f .
 - a (total) function: assigns a value to each number of A : denoted f
 - a (total) function²: $\forall x, \exists y | Rxy$: denoted f .
- Domain: The set of a functions arguments.
- Range: The set of its values. (Results)
- f is a function from a set A iff the domain of f is included in A

- f is a function to a set B iff its range is included in B .
- f^{-1} is the inverse of the function f from the set A to the set B iff: if for every member $b \in B$, there is exactly one member of $a \in A$ such that $f(a) = b$, then $f^{-1}(b) = a$, otherwise $f^{-1}(b)$ is undefined.
- f is onto B iff B is the range of f (Surjective)
Alt: (Wikipedia) : $\forall y \in Y, \exists x \in X | y = f(x)$
- f is one-to-one iff $\forall x \forall y (f(x) = f(y) \implies x = y)$ (Injective)
- f is a bijection iff f is onto and one-to-one.
- f is a correspondence iff f is total, one-to-one and onto.
- Sets A and B are equinumerous iff there is a correspondence from A to B .

Equinumerous is transitive. Prove: if A is equinumerous with B and B is equinumerous with C , then A is equinumerous with C . Proof: Suppose A is equinumerous to B , and B is equinumerous to C . Then: There is a total, one-to-one function f from A onto B , and a total one-to-one function g from B to C . Prove equinumerous via $h=g(f)$, such that $h(n)=g(f(n))$

- h is total: Let a be a member of A . $h(a) = g(f(a))$. Since f is total there is a member of b of B such that $f(a) = b$. since g is total, there is a member of $c \in C$ such that $g(b) = c$. Hence, h is total.
- h is onto C . WLOG Let c be a member of C , as g is onto, $\exists b \in B$ such that $g(b) = c$. As f is onto, then $\exists a \in A$ such that $f(a) = b$. Hence, the composition of $h = f(g)$ is onto C .
- h is one-to-one: Suppose h is not one-to-one.
Then there $\exists a_1, a_2 \in A$ such that $h(a_1) = h(a_2), a_1 \neq a_2$.
Giving $g(f(a_1)) = g(f(a_2)), a_1 \neq a_2$
Since g is one-to-one $g(b_1) = g(b_2)$ iff $b_1 = b_2$.
So the issue must lie in f . However f is one-to-one $f(a_1) = f(a_2)$ iff $f(a) = f(b)$. Which is a contradiction, giving us that h is one-to-one.

□

A^n : the n th Cartesian product of A with itself.

Suppose that the set of real Numbers $r, 0 < r < 1$, is enumerable. Then $L_r : r_1, r_2, r_3, \dots$ written in a notation of $0.n_1n_2n_3 \dots$ (n being natural numbers)

The set of functions from the set of positive integers to positive integers is not enumerable.

Proof. Suppose S is enumerable.

Then there is a list L_s of the members of S .

$L_s = \{s_1, s_2, s_3, \dots\}$

Let $\forall n \in \mathbb{N}, n \in k \iff n \notin S_n$

k is a set of positive integers.

so There is a number j such that $k = s_j$. So $j \in S_j \iff j \notin S_j$

Hence S is not enumerable.

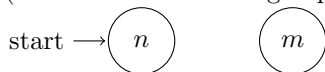
□

The set of total recursive functions from the set of positive integers, F^1 , is not enumerable.

It's a Proof by contradiction.

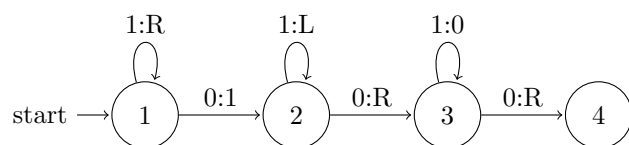
Turing machines are in the following form: $q_n, S_{1/0}, S_{1/0}/R/L, q_m$ where q_n is our current state, and you see $S_{1/0}$, perform function $S_{1/0}/R/$ and move to state q_m . If there is no operation specified on the current state for a scan, then it halts. (Also Called the Turing Alphabet)

Example with notation:



ex: (These are the same)

$$Q_1 S_1 R Q_1, Q_1 S_0 S_1 Q_2, Q_2 S_1 L Q_2, Q_2 S_0 R Q_3, Q_3 S_1 S_0 Q_3, Q_3 S_0 R Q_4$$



Remark (Turing Machines). • Each Turing machine is a finite set of Turing instructions.

- Each instruction is a 4 letter word of the Turing Alphabet.
- The set of Turing machine is enumerable. (Proof: exercise)

Definition (Standard initial configuration). A Turing machine is in a standard Initial configuration \iff

- for some positive integer k , there are k blocks of 1's on the tape.
- separated by a blank,
- and the rest of the tape is blank.
- the machine is scanning the left-most 1 on the tape.
- the machine is in it's lowest numbered state.

ex: $\dots 0010110111000\dots$ is a *SIC*. (if it's in lowest state) ex: $\dots 00010000\dots$ is a *SIC*.

Definition (Standard final configuration). A Turing machine is in a standard final configuration \iff

- there is a single block of k 1's
- and the rest of the tape is blank.
- the machine is scanning the left-most 1 on the tape.

ex: $\dots 00111111000\dots$ is a *SIC*. (if it's in lowest state) ex: $\dots 00010000\dots$ is a *SIC*.

Definition (Computes a one-place function f^1). A Turing M computes a one-place function f^1 : if M is started in a *SIC* with a single block of k 1's and

- if f^1 is defined for the argument k , then M eventually halts in a *SFC*
- or if f^1 is not defined for the argument k , then either M never halts or it halts in a non-standard final configuration.

Remark. Every Turing machine computes exactly one function of two arguments.

Remark. For any n , each Turing Machine computes exactly one function of n arguments.

The set of one-place Turing computable functions is enumerable

\vdots

The set of Turing computable functions is enumerable.

Definition (The halting problem). The problem of finding an effective method to determine whether a Turing machine will eventually halt or not after it is started with some input.

The halting problem is unsolvable. Ex: $L_M : M_1, \dots$

$h(m, n) =$

1 if M_m eventually halts after starting with input n

2 if M_m never halts after starting with input n

The halting problem is solvable iff h is computable. Show: h is not Turing computable.

Let C be a copying machine.

Let F be $\frac{1}{2}$ flipper.

Suppose h is Turing Computable.

Let H be a Turing machine that computes h .

If h is a Turing computable, then H exists.

If H exists, then $D(C - H - F)$ exists.

Let $D = M_k$, for some k . $M_k \in L_M$.

Start D with input k . The C -part of D will produce a copy of k , Then the H -part will do its job:

- If M_k will eventually halt after starting with input k , then H will produce output 1.
- If M_k will never halt after starting with input k , then H will produce output 2.

Then the F -part will do its job.

- If output from H is 1, F will never halt.
- If output from H is 2, F will eventually halt.

Giving us:

- If M_k will eventually halt after starting with input k , then D will never halt after starting with input k .
- If M_k will never halt after starting with input k , then D will eventually halt after starting with input k .

So M_k will halt, after starting with input k , \iff D will never halt after starting with input k .

Then M_k isn't identical with D , which is a contradiction! Hence D doesn't exist. So H does not exist. So h is not Turing computable. \square

Another halting problem..? $L_M : M_1, \dots$ $L_F : F_1, \dots$

$g(n) = 1$, if $f_n(n) = 2$

$g(n) = 2$, otherwise.

$g \neq f_k \forall k$

$h(m, n) = 1$ if M_m eventually halts after starting with input n .

$h(m, n) = 2$ if M_m never halts after starting with input n .

$s(m) = 1$, if M_m eventually halts after starting with input m .

$s(m) = 2$, if M_m never halts after starting with input m .

1. The halting problem is solvable iff h is computable.
2. If h is computable, then s is computable.
3. If s is computable, and TT 's is true, then g is computable.
4. g is not Turing computable.
5. Turing Thesis is true (Whatever is not Turing Computable is not computable)

6. The halting problem is not solvable.

3. Suppose S is computable and TT is true.

Then: There is a Turing machine S^* that computes s .

Suppose that we are to calculate $g(n)$, for some n .

Start S^* with input n .

- Case 1: S^* eventually halts with output 1

We know that M_n will eventually halt after it is started with input n . Start M_n with input n , when it halts, inspect the tape.

– Case 1.1: Halted in SFC $f_n(n) = 2$ $g(n) = 1$

– Case 1.2: Halted in non-SFC: f_n is undefined. $f_n(n) \neq 2$

And then Blake broke it:

As it's a halting problem to figure out if it's in *SFC*?

$g(n) = 2$

- Case 2: S^* eventually halts with output 2 We know that M_n will never halt after it is started with input n .

So we know that f_n is undefined for the argument n .

So we know that $g(n) = 2$

□

□