

CS 3313

Foundations of Computing:

1. Set theory review (from Discrete) ..
needed for remaining topics
2. Multitape TM Example(s)

<http://gw-cs3313.github.io>

(Discrete) Math Review

- Encoding integers...and enumeration (ordering)
- Cardinality of sets
 - Countable and Uncountable Sets
- Diagonalization technique (proof of contradiction)...next week labs
 - You need to understand this to understand unsolvable/undecidable problems

Integers, Strings, and Other data

- Data types are important as a programming tool.
 - Program manipulates a data type...operations are defined on data types
- But at another level, there is only one type, which you may think of as integers or strings.
 - A string of 0' and 1's !
- *Key point: Strings that are programs are just another way to think about the same one data type.*
- Recall data types and encodings from Architecture course.....

Example: Text

- Strings of ASCII or Unicode characters can be thought of as binary strings, with 8 or 16 bits/character.
- Binary strings can be thought of as integers.

Enumeration..i.e., ordering of strings

- Enumeration/ordering: It makes sense to talk about “the *i*-th binary string.”
 - Goal: to list all binary strings in some order
 - So we have the first string, second string, ... n-th string, etc...
- Consider set of all strings over $\{0,1\}$
 - $\{ \lambda, 0, 1, 00, 10, 000, \dots \}$
- Is the ordering just the decimal equivalent
 - Using the weighted positional binary representation of a decimal number ?
 - 101 is the number 5, etc.

Enumeration: Binary Strings to Integers

- There's a small glitch:
 - If you think simply of binary integers, then strings like 101, 0101, 00101,... all appear to be “the fifth string” since their decimal equivalent is the integer 5
- Fix by prepending a “1” to the string before converting to an integer (decimal equivalent).
 - Thus, 101 \rightarrow 1101, and 0101 \rightarrow 10101, and 00101 \rightarrow 100101
 - And therefore 101, 0101, and 00101 are the 13th, 21st, and 37th strings, respectively.
- $\{\lambda, 0, 1, 00, 01, \dots\}$ become $\{1, 10, 11, 100, \dots\}$
 - λ is first string, 0 is second string, 1 is third string,.....

Example: Images

- Represent an image in (say) GIF.
- The GIF file is an ASCII string.
- Convert string to binary.
- Convert binary string to integer.
- Now we have a notion of “*the i -th image.*”

Enumeration – Example/Question

- Consider base k numbers
 - $k=2$ is binary, $k=10$ is decimal, $k=16$ is Hex, etc.
- How would you generate/enumerate base k numbers of length n ?
- Ex.: if $k=2$ and $n=4$ then how would you enumerate the 4 bit numbers ?
- Ex: if $k=10$ and $n=2$ then how would you enumerate the 2-digit numbers
- Ex: if $k=3$ (symbols are 0,1,2) and $n= 3$ then...
- Question: can you generalize this as a procedure/algorithm?

Example: Enumerations of Proofs



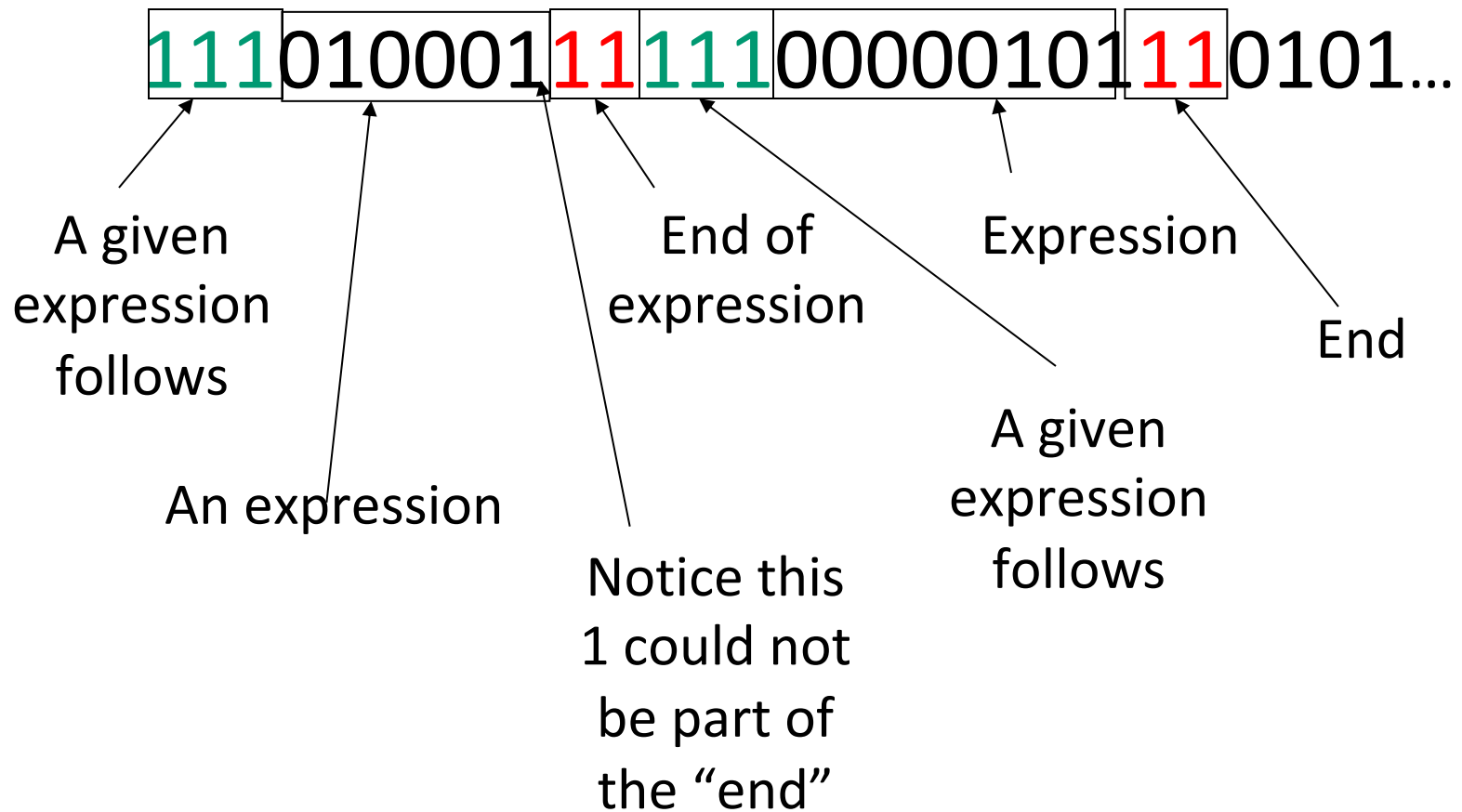
- A formal proof is a sequence of logical expressions, each of which follows from the ones before it....therefore:
- 1. Encode mathematical expressions of any kind in Unicode.
- 2. Convert expression to a binary string and then an integer.

Proofs – (2)

- But a proof is a sequence of expressions, so we need a way to *separate* them.
- Also, we need to indicate which expressions are given and which follow from previous expressions.
- Quick-and-dirty way to introduce new symbols into binary strings:
 1. Given a binary string, precede each bit by 0.
 - Example: 101 becomes 010001.
 2. Use strings of two or more 1's as the special symbols.
 - Example: 111 = “the following expression is given”;
 - Example: 11 = “end of expression.”

Key takeaway: remember this concept of using a specific pattern/string as a separator between fields

Example: Encoding Proofs



Example: Programs

- Programs are just another kind of data.
- Represent a program in ASCII.
- Convert to a binary string, then to an integer.
- Thus, it makes sense to talk about “the *i*-th program.”
- *Hmm...There aren't all that many programs.*

What's our takeaway?

- Languages, datatypes, programs, proofs can be encoded (as 0's and 1's) and *enumerated*
 - We can talk of the *i-th* program or the *i-th* proof etc.
- Next we formalize the notion of **counting/enumeration**....and the size of sets

Next: concept of Countable sets

- Arguments about the size of a set
- Diagonalization technique – to prove (by contradiction) .. Next weeks lab
- Why review this.....
- When we get to discussing properties of computability – specifically, undecidability (unsolvable problems), we need diagonalization technique to prove a problem is undecidable

Set Cardinality

- Cardinality of a set is the number of elements in the set
 - Size of the set
- Set can be finite or infinite
- Two sets A, B have the same cardinality if there is a one-to-one correspondence (mapping) from A to B
- $A = \{0, 1, 2, 3, 4, 5\}$ and $B = \{a, b, c, d, e, f\}$
 - $f(0)=a, f(1)=b, f(2)=c, f(3)=d, f(4)=e, f(5)=f$

Countable and Uncountable Sets

- Intuition: if we can arrange the elements of set in a manner where we can speak of “first element”, “second element”, etc.
- An infinite set A is **countably infinite** *if and only if* it has the same cardinality as the set of Natural numbers (positive integers)
 - There is a one to one correspondence (one to one and onto) from A to N .
- A set is **countable** *iff* it is finite or is countably infinite
- A set that is not countable is said to be **uncountable**
- Useful Theorems:
 - 1. If $A \subseteq B$ and B is countable then A is countable
 - 2. If $A \subseteq B$ and A is uncountable then B is uncountable

Enumerations

- An *enumeration* of a set is a 1-1 correspondence between the set and the positive integers.
- Thus, we have seen enumerations for strings, programs, proofs, and pairs of integers.

Countably Infinite Sets...Example

- Two sets have the same cardinality if there is a one to one correspondence between them
- Set of all integers Z is a countably infinite set: $f: Z \rightarrow N$
 - $f(0) = 1$ $f(-i) = 2i$ $f(+i) = 2i+1$
 - $0 \rightarrow 1, -1 \rightarrow 2, 1 \rightarrow 3, -2 \rightarrow 4, \dots$
 - “*ordering*” $0, -1, 1, -2, 2, -3, 3, \dots$
- Set of even integers Z_2 is countably infinite $f_2: N \rightarrow Z_2$
 - $f_2(n) = 2n$
- Set of primes P is countably infinite – f_3
 - $f_3(p) : p \text{ is the } i\text{-th prime.}$
 - Recall: we proved earlier that set of primes is infinite

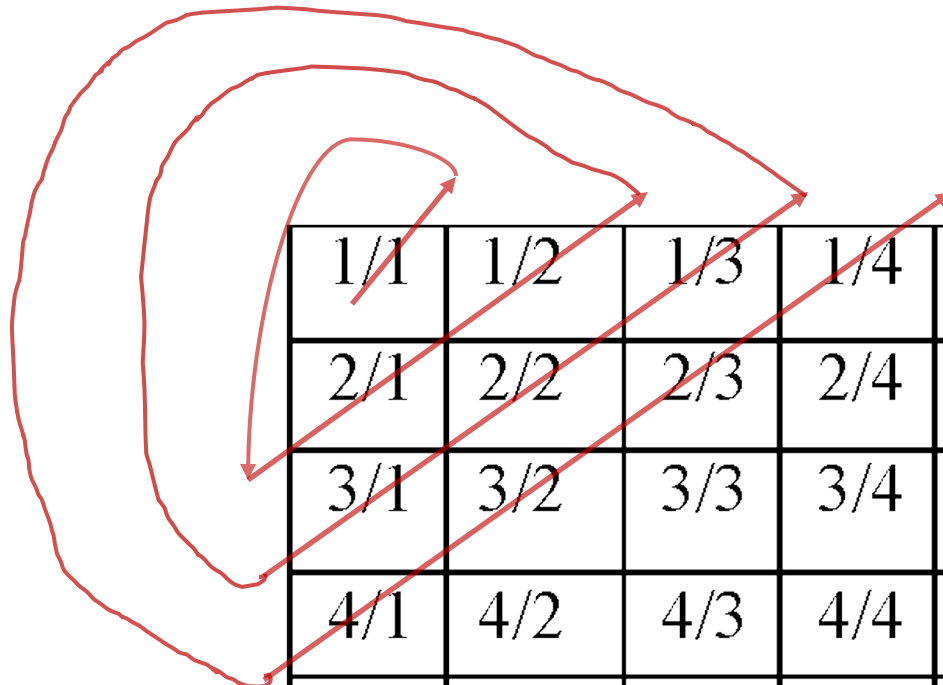
Rational Numbers \mathbb{Q}

- Rational number p/q p, q are integers
- Theorem: Set of positive rational numbers \mathbb{Q} is countable
- Intuition: list the rational numbers “in order”
 - Find a way to “label” the rational numbers to get the first rational number, the second rational number, etc.
 - For simplicity, let’s work with p, q positive
 - Observe: We can view the number p/q as a pair of integers $[p, q]$ and then order them first by sum and then by first component
 - $[1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2], \dots [1,4], [5,1] \dots$

Ordering of (positive) Rational Numbers \mathbb{Q}

- Make an infinite matrix containing all positive rationals.
 - The i -th row has all rational numbers with i in the numerator
 - The j -th column has all rational numbers with j in the denominator
 - Next turn this matrix into a list (ordered list)
- A bad way to do this would be to begin the list with all elements in the first row
 - Is not a good approach because first row is infinite and the list would never get to the second row!
- Instead, we list the elements on the diagonals
 - First element contain $1/1$, i.e., p/q where $p+q=2$
 - Second diagonal contains numbers p/q where $p+q=3$
 - Third diagonal contains numbers p/q where $p+q=4$
 -

Ordering of (positive) Rational Numbers \mathbb{Q}



1/1	1/2	1/3	1/4	1/5
2/1	2/2	2/3	2/4	2/5
3/1	3/2	3/3	3/4	3/5
4/1	4/2	4/3	4/4	4/5
5/1	5/2	5/3	5/4	5/5
....
...
...	...						

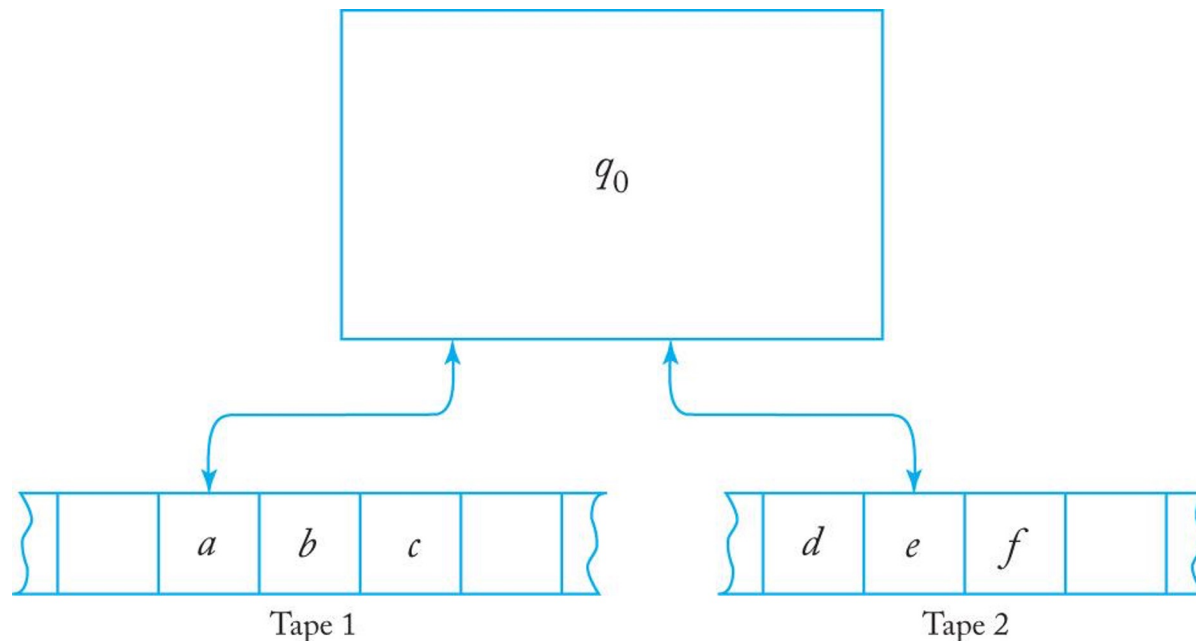
Rational Numbers \mathbb{Q}

- Deciphering the ordering...
- Observe: We can view the number p/q as a pair of integers $[p, q]$ and then order them first by sum and then by first component
 - $[1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2], \dots [1,4], [5,1] \dots$
 - Known as *filing order* for a 2-tuple

Multitape Turing Machines

- a *multitape Turing machine* has several tapes, each with its own independent read-write head
- A sample transition rule for a two-tape machine must consider the current symbols on both tapes:

$$\delta(q_0, a, e) = (q_1, x, y, L, R)$$



Questions/Clarifications on Multitape TMs ?

Cool example: Name-Value (key-value) Stores on a Turing Machine

- Name-Value or Key-value is a pair (n_i, v_i)
 - n_i is name/key/address
 - v_i is value associated with the name/key, i.e., value at that address
- Given key n_k find the value v_k
- Why bother with this.....
 1. Data(base) storage....from DB course a NoSQL DB (MongoDB?) model of Key-Value Databases
 2. **model of Computer memory**...address of a location and content at that location!

Key (or Address)	Value (content)
0	25
...	
34	200
...	...

Simulating a Name-Value (key-value) Store by a 2-tape TM

- Key-value is a pair ($n*v$); use separator symbols # between pairs
- The TM uses one of several tapes to hold an arbitrarily large sequence of name-value pairs in the format *#name*value#...*
- Mark (X), using a second track, the left end of the sequence.
- A second tape can hold a name whose value we want to look up.

Tape 1	# 0 * 25 # 1 * 33 #.....# 34 * 200 #.....
	X

Tape 2	34
-----------	----

Find value at “address” 34

Lookup

- Starting at the left end of the store (Tape 1), compare the lookup name with each name in the store.
 - Search for $\#n_k^*$
 - Ex: search for $\#34^*$
- When we find a match, take what follows between the $*$ and the next $\#$ as the value.
 - $\#n_k^* \textcolor{red}{v}_k \#$
 - Ex: 200 is the value with $n=34$
- Insert and Delete operations can be similarly supported.

Insertion

- Suppose we want to insert name-value pair (n, v) , or replace the current value associated with name n by v .
- Perform lookup for name n .
- If not found, add $n*v\#$ at the end of the store (Tape 1)
- If we find $\#n*v'\#$, we need to replace v' by v .
 - If v is shorter than v' , you can leave blanks to fill out the replacement.
 - **But** if v is longer than v' , you need to make room.
 - *Use the shifting over “subroutine”*

Insertion – (2)

- Use a third tape: copy everything from the first tape to the right of v' to Tape 3
- Mark the position of the $*$ to the left of v' before you do.
- On the first tape, write v just to the left of that star.
- Copy from the third tape to the first, leaving enough room for v .

Picture of Shifting Right

Example: $n=34$, $v' = 22$

Replace $v' = 22$ by $v = 1024$

Tape
1

... # 34 * 22 # 40* 27# 55*102#...

Tape
3

#40*27#55*102#...

Picture of Shifting Right

Example: $n=34$, $v' = 22$

Replace $v'=22$ by $v=1024$

Tape
1

... # 34 * 1024 blah blah...

Tape
3

#40*27#55*102#...

Picture of Shifting Right

Example: $n=34$, $v' = 22$

Replace $v'=22$ by $v=1024$

Tape
1

... # 34 * 1024 #40*27#55*102#...

Tape
3

#40*27#55*102#...

Questions on HW7 ?...Discussions