Learn the RAM model of computation to compare to bouchard and osbourne

* In RAM computing
  + Each step (+, -, = , \*, if, and, or, return from memory) takes one step
  + Memory is infinite, data registers can hold any size number?
  + Operations include Zero, Increment, Halt, and Decrement/jump (decrement zero means jump. This allows branching logic)
  + In RAM the instructions are ordered

turing\_computation.docx

To clearly define a complete models of computation

The following summaries were found searching for: see http://localhost/Archives/turing\_biology.docx

"turing complete" OR "universal turing machine" OR "turing equivalent" OR "Turing Completeness" OR "Turing equivalence" OR "computational universiality"

OR "church-turing thesis" AND biology

...from 2005 onward. For the biological journals listed at http://localhost/Archives/Biological\_Computation\_Community\_Feb\_03\_09.txt we went through all dates.

Turing Completeness see http://en.wikipedia.org/wiki/Turing\_complete

By computability theory, a turing complete computer can perform any calculation of any other computer model. Complete machines are physically impossible because they require unlimited memory, so this demand is usually overlooked when evaluating the universiality of a set of operations.

Most programming languages are turing complete, but there are much simpler examples, like automata, lambda calculus, formal grammer, formal langauge, rewrite systems, and Post Turing machines.

!automata theory the study of automatons

!.left associative f x y = (f x) y (evaluated left to right)

!.unary function a function with only one input

!.lambda calculus left associative unary functions so that ((Lyx.y-covered x)chocolate)ants -> chocolate-covered ants

!.formal grammer Noam Chompsky's formal system of generative grammers: non-terminal symbols (N), terminal symbols (E), is the application

production rules (P = ( E U N )\* N( E U N )\* -> (E U N)\* ) and starting with a start symbol (S) in N and proceeding till only terminals

.formal langauge the product of a formal grammer

!.rewrite systems used to transform subterms in math formulas. ie. times(plus(11,9),plus(2,4)) could become times(20, plus(2,4))

!.Post Turing machines Post independently developed Turing's idea of recursion using a bitstring

(history, start, operation) At a point on a bitstring of infinate 0's and scattered 1's either change values and move in direction Ji or change directions or stop

!.automatons (type, output) a finite state machine model which reads the symbols of a string and changes states. Its final state, is its output (y/n)

!.finite state machine Alphabet (E), states (S), initial state (s0), state transition function ( g: S x E -> S ), final states (F)

!kleen star \* the set of all strings over a symbol set

Theory of Computation http://en.wikipedia.org/wiki/Theory\_of\_computation

Divided into computability theory and complexity theory, computation theory explores the use of computation models for the solution of problems. Models include, Cominatory logic, mu-recursive functions, Markov algorithms, Register machines, and P". Other simpler models that are occasionally used include Regular expressions, finite automata, context-free grammars, pushdown automata, and primitive recursive functions.

!.concatenation (ie) concatentate('abc', 'zyx') == 'abczyx'

!.cartesian product (def, ie) all possible ordered pairs of two sets. ie. a = [1, 2], b = [3, 4], a x b = [(1, 3), (1, 4), (2, 3), (2, 4)]

!.computability theory (def, ie) determines if problems are computable, usually by deciding if strings are memebers of a 'lanugage', ie. is a number prime

!complexity theory focuses on the spatial and temporal domains required to solve a problem, introduction big(O) notation

!Combinatory logic a variant of lambda calculus in which lambda expression are replaced by a limited set of combinators

!.constant function returns the same value regardless of the input c(1) = c(2) = 5

!.successor function returns the next value in a series s(10) = 11

!.projection function (def) returns one of the arguments delivered p1(a, b, c) = a

!.composition operator (def, ie) feeds the results of one function into another. if a(1) = 10 and b(10) = 12 then (a comp b)(1) = 12

!.mu operator (def) searches for the minimum natural number fitting certain criteria

!.mu-recursive functions input a finite tuple of natural numbers to a partial functions and return a natural number using

(input, operations-5, output) constant function, successor function, projection function, Composition Operator, and mu operator

!.Markov algorithms string rewriting system that applies the first rule to the leftmost instance in the string and continues until termination.

(input, order of application) ie 1.) "|0" -> "0||", 2.) "1" -> "0|", 3.) "0" -> "" translates binary strings to their unary equvilant. PS. randomize application

!.Register machines (organization) cut the tape of Turing machines and run the gambit from theoretical to real, simple to universal

!.finite automata finite state machine with transitions between states according to rules. Actions may be performed when entering/leaving states.

(q, E, T, q0, F) states (Q), alphabet (E), Transtion Function ( Q x E -> Q), initial state (q0), accepting states (F) (F in Q)

!.singleton a set with exactly one element

!.regular languages (conditions - 3, !ie) 1. empty set or string, 2. all singletons, and 3. A U B, A.B, and A\* if A and B are regular,

(a^n b^n | n > 0) is not regular. Because 'ab' will be the same state as 'a'.'a'\*n.'b' where n is the number of states

!.Regular expressions (acceptors, chompsky) Express regular languages. Accepted by finite state automa or type 3 grammars from the chompsky hierarchy

!.context free left hand side of the equation is a singleton

!.context-free grammars (form, ie) V -> w where V is a non-terminal singleton - ie. S -> a | bS | aS describes all ab strings that end in a

!.pushdown automata (similar to, plus) like a finite automa with a stack that can be manipulated by state changes and serves as a form of memory

!.primitive recursive functions (form, ie-2) a function from natural numbers (N) to N, nearly all functions are primitively recursive (addition, division, etc)

combinators (?) ???

!.partial functions (rule) elements of the domain have either one or no associations to the codomain

!.natural number (def, alternative) n - {1, 2, 3} may include {0}

!.Associativity (def, ie) for all a, b, c in M: a\*(b\*c) = (a \* b) \* c

!.identity element (def, ie) there is element e in M such that for all a in M: a\*e = e\*a = a

!.binary map (def) association of elements of one set to another

!.binary operation (def) A binary map between the cartesian prodcuct of S (S x S) and S

Computability Theory

Answers the question of what is computable by which models. The models of computation proceed from finite state machines to pushdown automatons to turing machines. All work with strings and states. Pushdown automa add the capacity to write an arbitrary string which serves as memory, which turing machines are able to rewrite the string they are working on and therefor have both memory and recursion.

multi vs one tape turing (potential) there is no difference in computations that can be perfomed, though speed may be affected

Turing's Paper see http://www.abelard.org/turpap2/tp2-ie.asp#section-1

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM (1936)

Though this paper focuses on computable numbers (calculable using unlimited repetion of limited steps), it can be shown that these numbers are equivilant to computable functions and that both are enumarable (though the set of real numbers is not). At any particular time, the Turing Machine is only aware of the symbol it is scanning and it's state.

Our take home point here is that any computation can be performed using the following: An infininite string of finite letters, a finite stack of states, and a 'reader' which is aware of only a state and letter and moves or rewrites based on this combination. The idea of states is only to make for the simplest possible model. A turing machine could easily be built if it could read two points of the string simultanously as opposed to a point on the string and a state.

Automatic Machine One who's operation is purely dependent on internal configurations, rather than external inputs

Universal Turing Machine http://en.wikipedia.org/wiki/Universal\_Turing\_machine

Because the behavior table of a turing machine can be encoded in the input string, it is provable that a sufficiently large turing machine may encode the operations table or any other turing machine as a string in its code. Turing's model uses empty squares next to each output square to note the current operation and state.

List of Important/Influential/Introductory texts in CS http://en.wikipedia.org/wiki/List\_of\_important\_publications\_in\_computer\_science#Computability

Computation: (section for paper)

Because this paper is intended for a wide audiance (biological and computational), it is necessary to give a loose overview of the essential mechanisms of computation. Those interested in pursuing the proofs of these arguments might start at the wikipedia article and proceed to Turing's blah blah paper.

The theory of computation was formalized in 1936 by Alan Turing with his description of a universal computer. He proved (as did Church) that the actions of any computer could be performed by a system consisting of symbols written on an infinitely long tape upon which rests a read-head. The read-head may move, rewrite symbols on the tape and change states based upon the symbol it is on top of and its state.

To understand this method of computation it is useful to proceed from analogy. Turing's method of computation is inspired by simple calculations performed with paper and pencil. For example the method of addition learned in grade school may be modeled using Turing's Machine. For the purpose of terseness this example will reduce the carryover operation into two steps and only loosely define position behaviors. (see appendix for an expanded and complete system).

The 'tape' in this analogy is the paper, the alphabet is the numbers 1-9 and the + sign. The read head is our eyes which proceed from right to left and top to bottom. The states might be "add0' - 'add9' and the operations (based on symbol and state) are defined by the following table:

symbol 0 1 ... 9

State

add0 0, e 1, e ... 9, e

add1 1, e 2, e ... 10, e

... ... ... ... ...

add9 9, e 10, e ... 18, e

e add0 add1 ... add9

Using this process, one might add two arbitrarily large numbers. For example:

0.) state is 'e'

starting values 19 1.) read '9', state becomes 'add9' 3.) read '1', state becomes 'add1'

+ 85 2.) read '5', print '16' (below) and state becomes 'e' 4.) read '8', print '9' (below) state becomes 'e'

summand of 9+5 16 5.) read 6, state becomes 'add6' 7.) read '1', state becomes 'add1'

summand of 1+8\* + 90 6.) read 0, print '6', state becomes 'e' 8.) read '1', state becomes 'add1'

solution 106 tada

\*note the 0 behind the 9 is not accounted for by this model because of the complexity, loosely we might say that the position of the summands is preservered, but for a more formal treatment please see the appendix.

Even this rudimentary example can be difficult to look at. The important point to note is that with only a 100 operational rules, an 11 symbol alphabet and a narrow focus on one value and one state at a time, the read head is capable of adding arbitrarily long numbers. Subtraction, division and so on might necessitate a different set of rules. Rather than changing the design of the computer, Turing showed that these new rules can be inscribed into the initial input string. By this method turing shows that a relatively simple computer model can be made that simulates the behavior of any computer model.

This finding, astounding in 1936, has important implications for modern computing and biological computing. Because computers can emulate one another, changes can be made to a computer's hardware without affecting the software that runs on top of it. This abstraction between program and computer means that biological systems which fit Turing's model for computation have the potential of running any program currently or inevitiably concieved.

That is not to say that this conversion will be straightforward. Biology offers a number of challenges to the development of a computational interface. While Turing's model and modern computers are intollerent of look-up errors, biological systems are likely to be probabalistic and hence error prone. Despite the evidence established in this paper there is little doubt that we are not completely aware of all of the naturally occuring biological operations which will complicate program design. Last, we do not have sufficient input/output methods to biological systems to justify thier superiority to silicon models in any case. However, the pursuit of such IOs is an important step in the modeling of biological systems and ultimately the capitalization of thier computational capacity both to facilliatate human computatation and affect cellular behavior.

Note: A system can only be Turing complete if the initial conditions needed to perform a calculation can be supplied by a machine which is not complete http://www.wolframscience.com/prizes/tm23/TM23Proof.pdf

General intro to computing - free - http://www.cse.ohio-state.edu/~gurari/theory-bk/theory-bk.html

more computability logic - http://www.cis.upenn.edu/~giorgi/cl.html#2