The Church-Turing Thesis

Chapter 18

Are We Done?

FSM ⇒ PDA ⇒ Turing machine

Is this the end of the line?

There are still problems we cannot solve:

- There is a countably infinite number of Turing machines since we can lexicographically enumerate all the strings that correspond to syntactically legal Turing machines.
- There is an uncountably infinite number of languages over any nonempty alphabet.
- So there are more languages than there are Turing machines.

What Can Algorithms Do?

1. Can we make all true statements theorems?
2. Can we decide whether a statement is a theorem?

The Entscheidungsproblem

David Hilbert (1928)

Does there exist an algorithm to decide, given an arbitrary sentence *w* in first order logic, whether *w* is valid?

Given a set of axioms *A* and a sentence *w*, does there exist an algorithm to decide whether *w* is entailed by *A*?

Given a set of axioms, *A*, and a sentence, *w*, does there exist an algorithm to decide whether *w* can be proved from *A*?

This study source was downloaded by 100000831946134 from CourseHero.com on 09-25-2022 22:53:37 GMT -05:00

Gödel’s Incompleteness Theorem

Kurt Gödel showed, in the proof of his Incompleteness Theorem [Gödel 1931], that the answer to question 1 is no. In particular, he showed that there exists no decidable axiomatization of Peano arithmetic that is both consistent and complete.

Negative answer – Church, Turing

To answer the question, in any of these forms, requires formalizing the definition of an algorithm:

- Church: lambda calculus - 1935
- Turing: Turing machines – 1936

Church proved that there is no computable function which decides for two given lambda calculus expressions whether they are equivalent or not

Turing reduced the halting problem for Turing machines to the Entscheidunsproblem

Turing proved that Turing machines and the lambda calculus are equivalent.

Church's Thesis
(Church-Turing Thesis)

All formalisms powerful enough to describe everything we think of as a computational algorithm are equivalent.

This isn’t a formal statement, so we can’t prove it. But many different computational models have been proposed and they all turn out to be equivalent.

The Church-Turing Thesis

Examples of equivalent formalisms:

- Modern computers (with unbounded memory)
- Lambda calculus
- Partial recursive functions
- Tag systems (FSM plus FIFO queue)
- Unrestricted grammars:

  $aSa \to B$
- Post production systems
- Markov algorithms
- Conway’s Game of Life
- One dimensional cellular automata

This study source was downloaded by 100000831946134 from CourseHero.com on 09-25-2022 22:53:37 GMT -05:00

The Lambda Calculus

The successor function:

$(\lambda\ x.\ x + 1)$

$(\lambda\ x.\ x + 1)\ 3 = 4$

Addition: $(\lambda\ x.\ \lambda\ y.\ x + y)\ 3\ 4$

This expression is evaluated by binding 3 to x to create the new function $(\lambda\ y.\ 3 + y)$, which is applied to 4 to return 7.

In the pure lambda calculus, there is no built in data type number. All expressions are functions. But the natural numbers can be defined as lambda calculus functions. So the lambda calculus can effectively describe numeric functions just as we have done.

Tag Systems

A tag system (or a Post machine) is an FSM augmented with a FIFO queue.

A tag system for WW:

A tag system for WW^R:

Tag systems are equivalent in power to Turing machines because the TM's tape can be simulated with the FIFO queue.

Idea: treat the queue as a loop

Post Production Systems

A ***Post system*** P is a quintuple (V, Σ, X, R, S):

- V is the rule alphabet,
- Σ is a subset of V,
- X is a set of variables with values in V^*,
- R (the set of rules) is a finite subset of:

  $(V \cup X)^* \times (V \cup X)^*$

  Every variable on the RHS must also be on the LHS.
- S can be any element of $V - \Sigma$.

- LHS may contain multiple symbols.
- Rules may contain variables.
- LHS must match entire string. So, $A \rightarrow B$ becomes:

$XAY \rightarrow XBY$

A Post Production System

$WW = \{ww : w \in \{\texttt{a}, \texttt{b}\}^*\}$.

$P = (\{S, \texttt{a}, \texttt{b}\}, \{\texttt{a}, \texttt{b}\}, \{X, Y\}, R, S)$, R =

(1) $XS \rightarrow X\texttt{a}S$ /* Generate $(\texttt{a} \cup \texttt{b})^*\ S$.
(2) $XS \rightarrow X\texttt{b}S$ "
(3) $XS \rightarrow XX$ /* Create a second copy of X.

This Post system can generate, for example, the string `abbabb`:

$S \Rightarrow$ (using (1) and letting X match ε)
$\texttt{a}S \Rightarrow$ (using (2) and letting X match `a`)
$\texttt{ab}S \Rightarrow$ (using (2) and letting X match `ab`)
$\texttt{abb}S \Rightarrow$ (using (3) and letting X match `abb`)
`abbabb`

This study source was downloaded by 100000831946134 from CourseHero.com on 09-25-2022 22:53:37 GMT -05:00

Markov Algorithms

A Markov algorithm *M* is a triple (*V*, Σ, *R*), where:

- *V* is the rule alphabet, which contains both working symbols and input symbols. If *M* is a decider or a semidecider, *V* will contain two special working symbols, *Accept* and *Reject*.
- Σ is a subset of *V*, and
- *R* is an ordered list of rules, each of which is an element of $V^* \times V^*$. There are two kinds of rules:
  - Continuing, written as $X \to Y$, and
  - Terminating, written as $X \to\bullet\ Y$

As with Turing machines, there is no special start symbol, but there will be an input string.

A Markov Algorithm Interpreter

```
Markovalgorithm(M, w) =
  1. Until no rules apply or the process has been terminated by
     executing a terminal rule do:
       1.1 Find the first rule in the list R that matches against w.
           If that rule matches w in more than one place, choose
           the leftmost match.
       1.2 If no rule matches then exit.
       1.3 Apply the matched rule to w by replacing the substring
           that matched the rule's LHS with the rule's RHS.
       1.4 If the matched rule is a terminating rule, exit.
  2. If w contains the symbol Accept then accept.
  3. If w contains the symbol Reject then reject.
  4. Otherwise, return w.
```

Notice: Markov algorithms are deterministic.

A Markov Algorithm for $A^nB^nC^n$

(1) `#a` → `%`
(2) `#b` → • *Reject*
(3) `#c` → • *Reject*
(4) `%a` → `a%`
(5) `%b` → `?`
(6) `%` → • *Reject*
(7) `?b` → `b?`
(8) `?c` → ε
(9) `?` → • *Reject*
(10) `#` → • *Accept*
(11) ε → `#`

Example: `aabbbccc`

The Power of Markov Algorithms

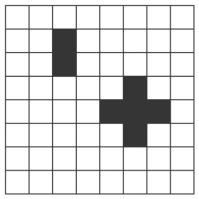
Theorem: The Markov algorithm formalism is equivalent in power to the Turing machine.

Proof: By two constructions. Show that each formalism can simulate the other.

This study source was downloaded by 100000831946134 from CourseHero.com on 09-25-2022 22:53:37 GMT -05:00

The Game of Life

Playing the game



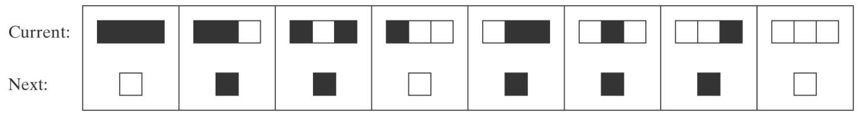
At each step of the computation, the value for each cell is determined by computing the number of neighbors (up to a max of 8) it currently has, according to the following rules:

- A dead cell with exactly three live neighbors becomes a live cell (birth).
- A live cell with two or three live neighbors stays alive (survival).
- In all other cases, a cell dies or remains dead (overcrowding or loneliness).

We'll say that a game halts iff it reaches some stable configuration.

Elementary Cellular Automata

Wolfram's Rule 110 is a universal computer if you can figure out how to encode the program and the input in the initial configuration:



This study source was downloaded by 100000831946134 from CourseHero.com on 09-25-2022 22:53:37 GMT -05:00