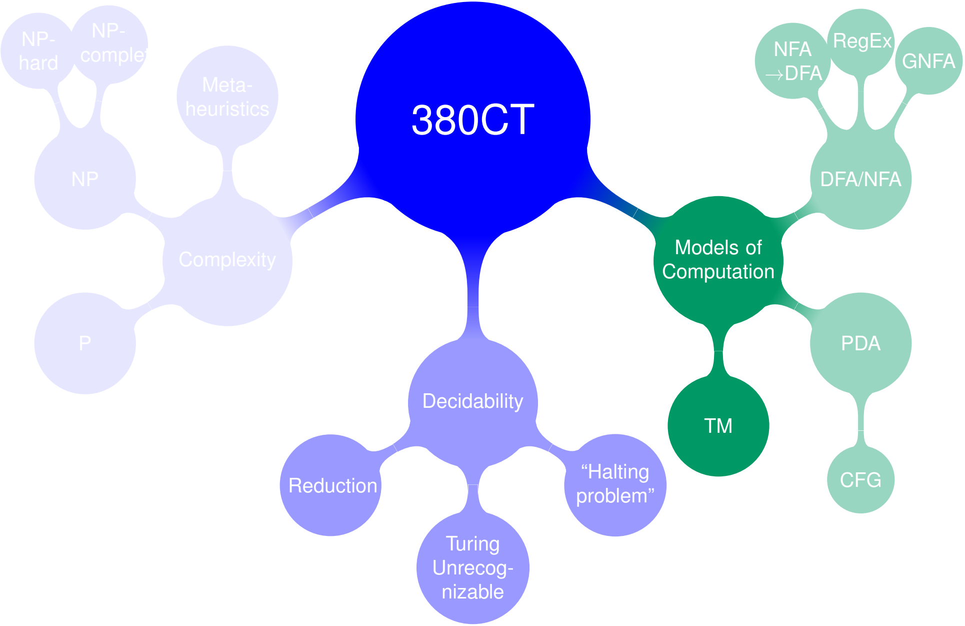
Turing Machines (TMs)

Dr Kamal Bentahar

School of Engineering, Environment and Computing
Coventry University

7/11/2016


380CT
Complexity
NP
NP-hard
NP-complete
Meta-heuristics
P
Decidability
Reduction
Turing Unrecognizable
"Halting problem"
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM

History: nature of computing

Questions about this first arose in the context of pure Mathematics:

- Gottlob Frege (1848–1925)
- David Hilbert (1862–1943)
- George Cantor (1845–1918)
- Kurt Gödel (1906–1978)
- 1936:
  - Gödel and Stephen Kleene (1909-1994): **Partial Recursive Functions**
  - Gödel, Kleene and Jacques Herbrand (1908–1931)
  - Alonzo Church (1903–1995): **Lambda Calculus**
  - Alan Turing (1912–1954): **Turing Machine**
- 1943: Emil Post (1897–1954): **Post Systems**
- 1954: A.A. Markov: Theory of Algorithms – **Grammars**
- 1963: Shepherdson and Sturgis: **Universal Register Machines**

The Church-Turing Thesis

- It turns out that the "Turing Machine model" and *all* the other models of general purpose computation that have been proposed are equivalent!
- They all share the essential feature of **unrestricted access to unlimited memory.**

  As opposed to the DFA/NFA/PDA models for example.
- They all satisfy reasonable requirements such as the ability to perform only a finite amount of work in a single step.
- They all can **simulate** each other!

Philosophical Corollary: Church-Turing Thesis

Every *effective computation* can be carried out by a TM.

i.e. *algorithmically computable* $\iff$ computable by a TM.

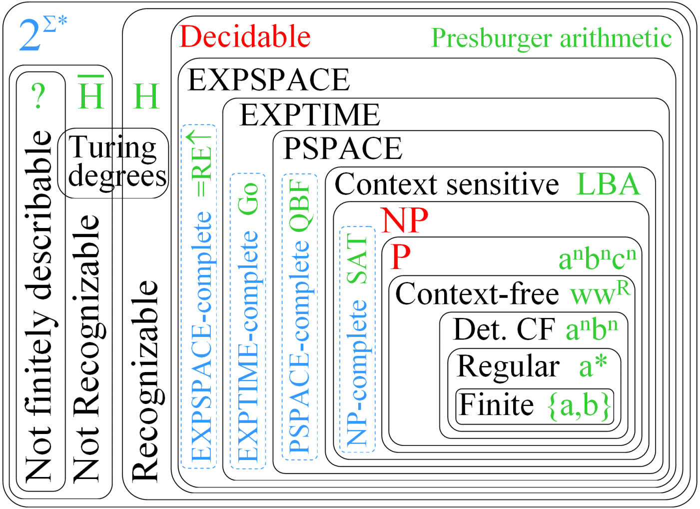
See http://plato.stanford.edu/entries/church-turing/ and http://en.wikipedia.org/wiki/Church-Turing_thesis for discussion.

Algorithms

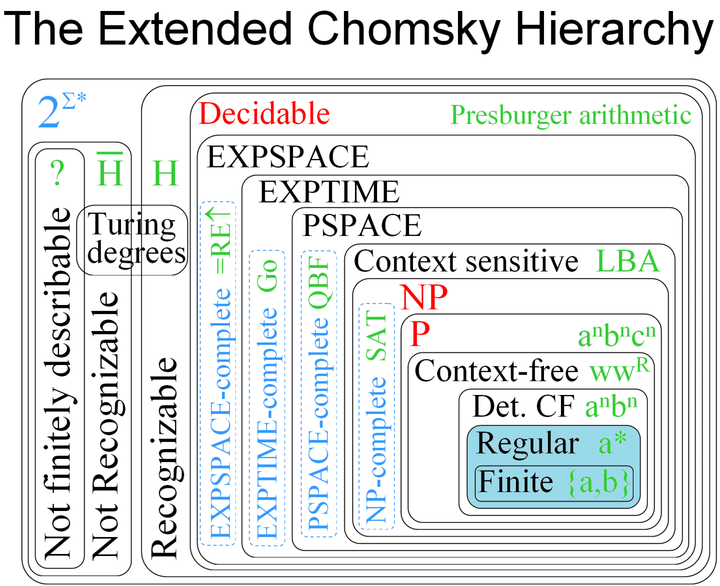
In a sense, the Church-Turing thesis implies that the underlying class of “algorithms” described by all these models of computation is the same, and corresponds to the natural intuitive concept of *algorithms*.

$$\text{Intuitive concept of algorithms} \quad = \quad \text{Turing machine algorithms}$$

The Extended Chomsky Hierarchy

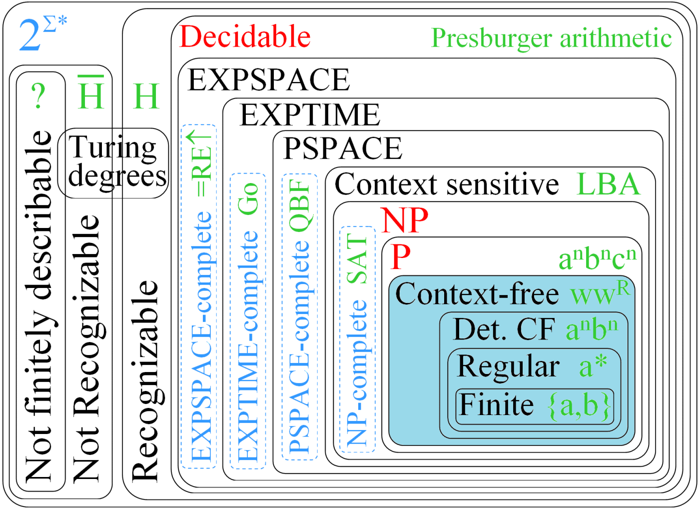


Chomsky Hierarchy

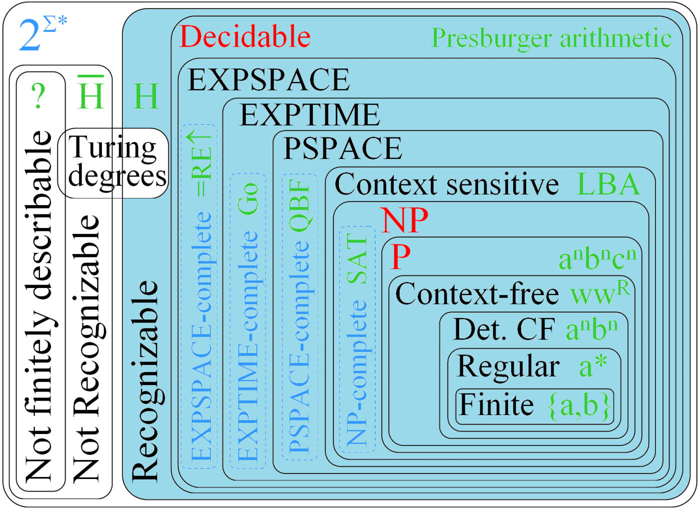


Chomsky Hierarchy

The Extended Chomsky Hierarchy



The Extended Chomsky Hierarchy



Chomsky Hierarchy

Grammar	Languages	Automaton	Production rules
Type-0	Recursively Enumerable	Turing Machine (TM)	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context Sensitive	Linear-bounded TM	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context Free	PDA	$A \rightarrow \gamma$
Type-3	Regular	NFA/DFA	$A \rightarrow aB \mid a$

$a, b, \ldots$ *Terminals – constitute the strings of the language*
$A, B, \ldots$ *Non-terminals – should be replaced*
$\alpha, \beta, \ldots$ *Combinations of the above*

TMs vs NFAs/PDAs

- TM are similar to NFA/PDA, but has access to unlimited memory.
- No known model of computation is more powerful than a TM.

The main differences are:

1. TM may store the entire input string and refer to it **as often as needed**.
2. Special states for **accepting** and **rejecting** which take **immediate effect**. (No need to reach the end of the input string.)
   The TM has the potential to go on with its computation for ever, without reaching either an accept or reject state $\rightarrow$ the **"Halting Problem"**.

Turing Machine

- A Turing machine has an **infinite tape** (memory).
- It has a **tape head**, which may **read** and **write** symbols and **move** around the tape.
- Initially: tape contains only the *input string*; *blank everywhere else*.
- If the machine needs to store information, it can write it on the tape.
- It has designated **accept** and **reject** states.
- The machine can only terminate on reaching one or the other; otherwise, it will just keep going. . . !
- The TM's **transition function** is defined according to the *state* of the machine and the *symbol currently being read* by the tape head.
- Given a **state and symbol** pair, the machine may **change state, write a symbol** onto the tape and **move left or right** by one space.

Turing Machine Computation

- The input is placed on the tape. The rest of the tape is blank.
- The head starts on the leftmost square of the input string.
- The computation proceeds according to the rules of δ.
- Computation continues until it enters either an accept or reject state.
- As the TM computes, changes occur in the current state, the current tape contents and the current head location.
- A set of these three items is called a **configuration**

Configuration may be represented in the form uqv, where

- u is the string of all symbols to the left of the head
- v is the string consisting of the symbol at the current head location, followed by all symbols to the right of it
- q is the current state

e.g. If the tape contents are 10010, the machine is in state q_6, and the head is over the second zero we write: $10q_6010$

Turing Machines (TMs) – languages

Decidable languages

A language is (Turing) **decidable** if some TM *decides* it.

i.e. given a string w:

- if w is **in** the language: the TM will **accept** it.
- it w is **not in** the language: the TM will **reject** it.
  Such TMs are called **deciders.**

Turing recognizable languages

A language is **Turing recognizable** if some TM *recognizes* it.

i.e. given a string w:

- if w is **in** the language: the TM will **accept** it.
- it w is **not in** the language then the TM may **reject** it or **never halt**.

Description of algorithms and TMs

Specification can be at one of 3 levels of detail:

1. Formal description (Transition diagrams, etc.).
2. Implementation description.
   (Describe how TM manages tape and moves head).
3. High-level description (Pseudocode or higher).

Also, we usually specify how to **encode** objects (if not “standard”), and the exact **input** and **output**.

Example (TM to recognize $\{w\#w \mid w = \{0,1\}^*\}$)

- Scan the input to check it contains only a single # symbol.
  If not, reject.
- Zig-zag across the tape to corresponding symbols on either side of the # symbol, crossing off each one.
  If they are not the same, reject.
- When all symbols to the left of the # are crossed off, check for remaining symbols to the right. If there are reject, otherwise accept.

Task: Trace the following inputs

$\texttt{01}\#\texttt{01}$ $\texttt{01}\#\texttt{011}$ $\texttt{011}\#\texttt{01}$ $\texttt{01}\#\#\texttt{01}$

Example (TM to recognize $\{0^{2^n} \mid n \geq 0\}$)

This language consists of all strings of 0's whose length is a power of 2.

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0 then accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0's was odd then reject.
4. Return to the left hand end of the tape.
5. Go to stage 1.

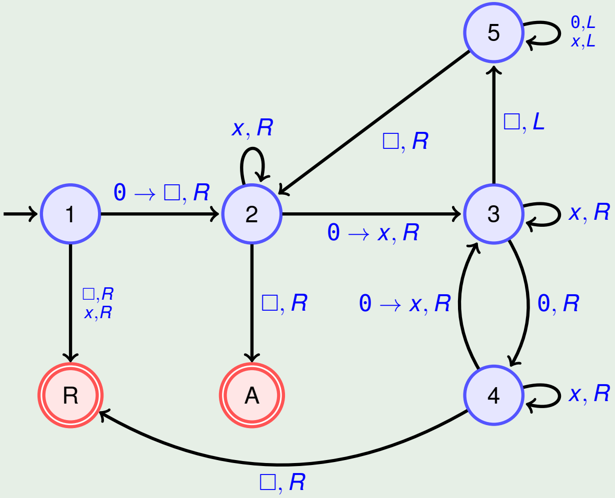
Task: Trace the following inputs

$$0, 0^2, 0^3, 0^4, 0^7$$

Example (TM to recognize $A = \{0^{2^n} \mid n \geq 0\}$)

Formal description:

- $Q = \{1, 2, 3, 4, 5, A, R\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \square\}$
- The start, accept and reject states are 1, A and R, respectively.
- δ is given by the state diagram:



Notation:
a → b, *R*: on reading a on the tape: replace it with b,then move to the right.
a, *R*: shorthand for a → a, *R*

Formal Definition of a TM

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ where

- Q is the finite set of states
- Σ is the input alphabet, not containing the special *blank symbol*: $\square$
- Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subset \Gamma$
- $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$ is the transition function
- q_{start} is the start state
- q_{accept} is the accept state
- q_{reject} is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$

Multi-tape TMs

- A **multi-tape TM**, is a TM with *more than one tape*.
- More transitions need to be defined, but it **simplifies computations**

Example ($\{w\#w \mid w = \{0,1\}^*\}$)

One-tape: zig-zag around $\#$ crossing off matching symbols. Requires two loops.

Multi-tape: write the second half in the second tape, then use a single loop to check it matches the first half.

Equivalence

Every multi-tape TM has an equivalent single-tape TM.

Nondeterministic Turing Machines (NTMs)

- A configuration can have **zero or more** subsequent configurations.
  $\rightarrow$ The machine may be in many configurations at the same time. Imagine the TM self-replicating as it goes along.

  Interestingly, the closest real thing we have to an NTM is **DNA computation**, as the processed units are artificially manufactured chromosomes (capable of self-replication). This still is not really nondeterministic as there is a finite limit to the number of DNA strands which may exist during computation.
- Subtlety: If a non-deterministic TM is a decider then **all** branches need to reject for it to **reject** a string.
- Deterministic and nondeterministic TMs recognize the same languages!

Equivalence

Every NTM has an equivalent deterministic TM.

Next week...

Limits of computation...

Even a TM cannot solve certain problems!

Such problems are beyond the theoretical limits of computation (unsolvable)