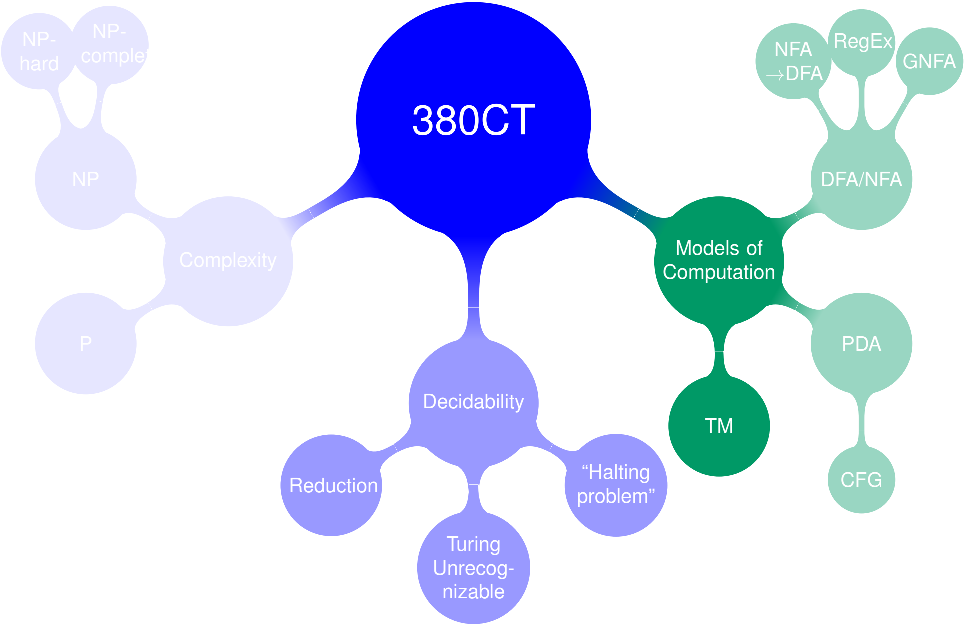
Turing Machines (TMs)

Dr Kamal Bentahar

School of Computing, Electronics and Mathematics
Coventry University

Lecture 6


380CT
Complexity
NP
P
NP-hard
NP-complet
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM
Decidability
Reduction
Turing Unrecog-nizable
“Halting problem”

Last week. . . Grammars/Chomsky Hierarchy

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

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

Turing Machine (TM)

- Similar to NFA/PDA, but has unrestricted access to unlimited memory.
- No known model of computation is more powerful than the TM model.

The main differences are:

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

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

1. Scan the input to check it contains only a single $\#$ symbol.
   If not then **reject**.
2. Zig-zag across the tape to corresponding symbols on either side of the $\#$ symbol, crossing off each matching pair.
   If they do not match then **reject**.
3. When all symbols to the left of the $\#$ are crossed off, check for any remaining symbols to the right. If there are then **reject**, otherwise **accept**.

Task: Trace the TM on the following inputs:

`01#01`   `011#01`   `01#011`   `01##01`

Turing Machine (TM)

- TM has an **infinite tape** (memory), divided into cells.
- It has a tape **head**, which may **read** and **write** symbols and **move** around.
- Initially: tape contains only the *input string* (blank everywhere else).
- If the TM needs to store information, it can write it on the tape.
- It has designated **accept** and **reject** states.

  Can only terminate on reaching one or the other; otherwise, it will just keep going!
- **Transition function** δ: Given a (*state*, *symbol*) pair, the TM will:
  1. change state,
  2. write a symbol (in the current cell)
  3. and move left or right (by one cell).

Turing Machine Computation

- Input is placed on tape; rest of the tape is blank.
- Head starts on the leftmost cell of the input.
- Computation proceeds according to the rules of δ.
- Computation continues until it enters either an **accept** or **reject** state.

Configuration – notation

The snapshot of the tape and head at a given time is called a **configuration**.

Notation: uqv

- u: string to left of head.
- q: current state.
- v: string to right of head including the current head location.

e.g. tape contains 10010, TM is in state q_6, and head is over the second zero
$\rightarrow$ write: $10q_6010$

Decidable and recognizable languages

Decidable languages

A language is **decidable** if some TM **decides** it.
Namely, given a string *w*:

- if *w* is **in** the language then the TM will **accept** it.
- it *w* is **not in** the language then the TM will **reject** it.

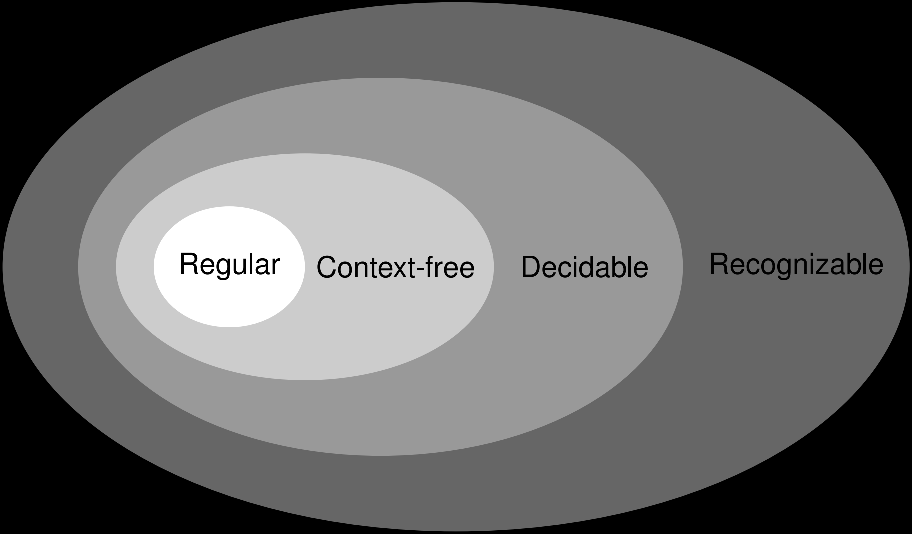
Such TMs are called **deciders.**

Recognizable languages

A language is **recognizable** if some TM **recognizes** it.
Namely, given a string *w*:

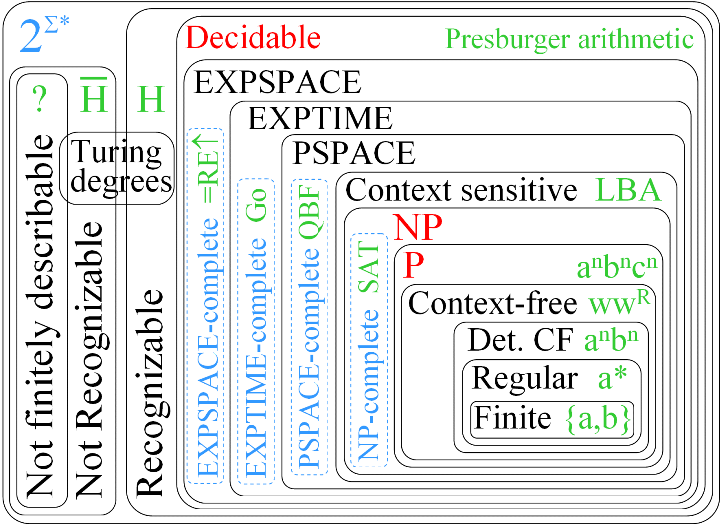
- if *w* is **in** the language: the TM will **accept** it.
- it *w* is **not in** the language then the TM may **reject** it or **never halt.**

The “computation universe” discovered so far...



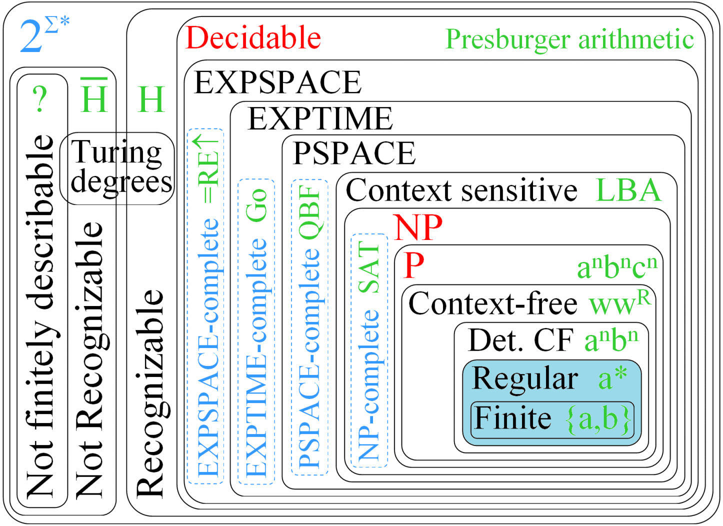
Chomsky Hierarchy

The Extended Chomsky Hierarchy



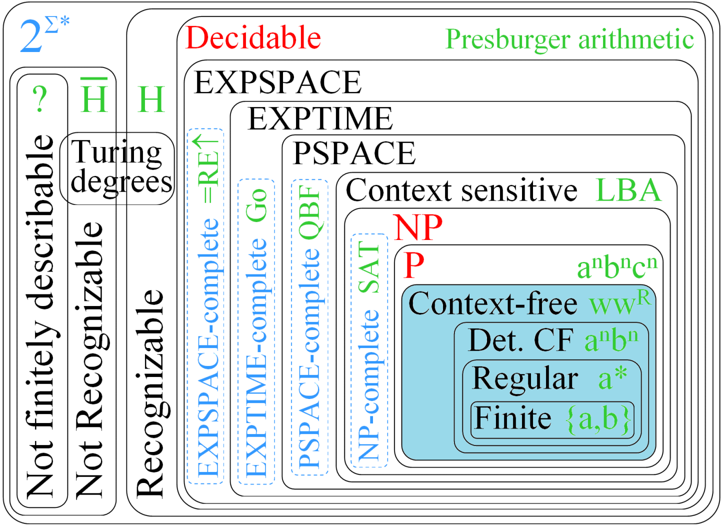
Chomsky Hierarchy

The Extended Chomsky Hierarchy

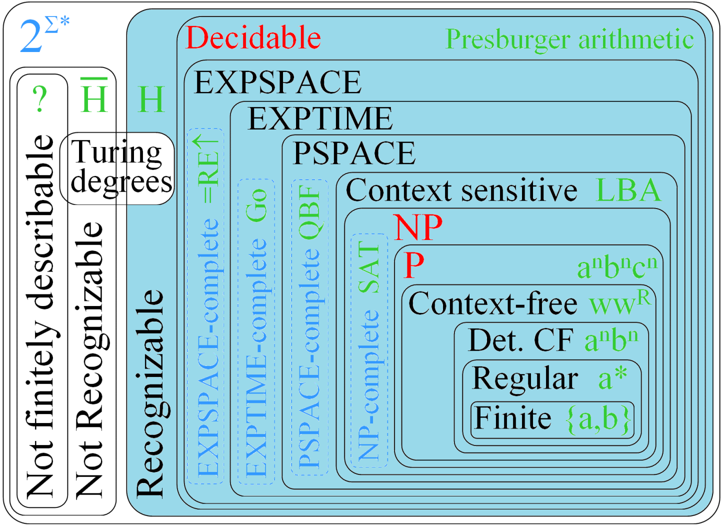


Chomsky Hierarchy

The Extended Chomsky Hierarchy



The Extended Chomsky Hierarchy



Description of algorithms and TMs

Three possible 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.

We also specify how to **encode** objects (if not obvious/standard), and the exact **input** and **output**.

Example ($L = \{0^{2^n} \mid n \geq 0\}$ (1/3) High level)

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

$$L = \{0, 00, 0000, 00000000, 0^{16}, 0^{32}, \ldots\}$$

Input: String $s \in \{0\}^+$.
Output: *true* if $|s|$ is a power of 2; *false* otherwise.

```
while |s| is even do
    s ← half of s
end while
if |s| = 1 then
    return true
else
    return false
end if
```

Example ($L = \{0^{2^n} \mid n \geq 0\}$ (2/3) Implementation level)

1. Scan left to right across the tape, crossing off every other 0.
2. If only a single 0 remains then **accept**.
3. If an odd number of 0's remain then **reject**.
4. Return to the left hand end of the tape.
5. Go to step 1.

Task: Trace the following inputs:

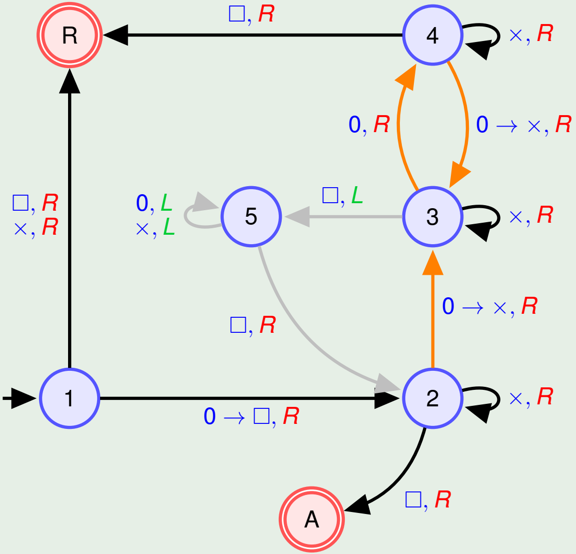
$$0, 0^2, 0^3, 0^4, 0^7$$

Example ($L = \{0^{2^n} \mid n \geq 0\}$ (3/3) Formal description)

Notation:
a → b, R: read a on the tape:
replace it with b,
then move to the right.
(L: left.)
a, R: shorthand for a → a, R

Formal description:

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



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 $q_{\text{accept}} \neq q_{\text{reject}}$

Nondeterministic TMs (NTMs/NDTMs)

- For an NTM, a given *configuration* can have **zero or more** subsequent configurations.
  $\rightarrow$ TM may be in many configurations at the same time.
  Imagine the NTM *self-replicating* as it goes along.
- If an NTM is a *decider* then:
  - it *accepts* as soon as **any branch accepts**,
  - it only *rejects* if **all its branches reject**.
- Deterministic and nondeterministic TMs recognize the same languages!

Equivalence

Every NTM has an equivalent deterministic TM.

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 nested 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.

TMs in real life

- The closest we have to an NTM is **DNA computation**:
  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.
- **Quantum computers** promise to be faster than the classical-physics machines that we currently have, but they are still equivalent to Turing Machines.

What is computation?

A *computational procedure* is called **effective** if:

- it is set out in terms of a finite number of exact instructions,
- it will produce the desired result in a finite number of steps,
- in principle, it can be carried out by a human being, unaided by any machinery except paper and pencil,
- it demands no insight, intuition, or ingenuity, on the part of the human carrying out the procedure.

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**

Equivalence of all of these models $\rightarrow$ "Church-Turing Thesis"

All these models define exactly the same class of computable functions!
$\rightarrow$ Anything that is computable can be computed by some TM.

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 one essential feature: **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* $\Longleftrightarrow$ 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 class of “algorithms” described by all these models of computation is the same, and corresponds to the natural *intuitive concept of algorithms*.

Using TMs to formallly define “algorithms”

Intuitive concept of algorithms = Turing machine algorithms

Next week...

Limits of computation...

Even TMs **cannot** solve all problems!

There are problems that are beyond the theoretical limits of computation!