Monash University
Faculty of Information Technology

Lecture 19
Polynomial time, and the class P

Slides by Graham Farr (2012), with minor modifications (2013).

FIT2014 Theory of Computation

Overview

- Decidability: it's not enough
- Time complexity
- Polynomial time
- The class P
- Properties of P
- Examples of languages in P

Decidable languages

- Solvable, in principle, by a computer
- Doesn't matter (much) which definition of "*computer*" you use.

  The set of ***decidable languages*** is the same.

- BUT what about the resources required? Which languages (or problems) can be solved efficiently, in practice?
- The most important resource: ***time***

Time complexity

- The time taken by a Turing machine M on an input x is the number of steps M takes until it halts.
- Time complexity:

  $t_M(n)$ = maximum time taken by M for any input of length n

- It's a function of n, which can be any positive integer. For it to be defined, M must halt on all inputs.
- *Worst case*

Time complexity

- Example: Turing machine to decide whether a string ends in b
  - moves to the right until reaches first blank symbol, then does one step to the left and accepts if the symbol there is b, otherwise rejects
  - time complexity? $n + 1$
- Example: Turing machine to decide whether a string is a palindrome
  - time complexity? $\frac{1}{2} . n(n+2)$
- Example: Turing machine to decide whether a string is empty
  - time complexity? 1

Time complexity

- Exercise: consider any regular language.
- What can you say about its time complexity?

- See Lecture 14 on Turing Machines:

  finite automaton $\rightarrow$ Turing machine

Time complexity

- Depends on the type of Turing machine (or computer) used.
- Some details that affect time taken:

  number of symbols used, whether tape is infinite in both directions (or just one), number of tapes, dimensionality of "tape" (1-D, 2-D, …?), whether allowed movement directions include sitting still, …
- In FIT1029/1045 and FIT1008, you met time complexity of algorithms and programs.

  Those time complexities still assumed some theoretical computer (maybe implicitly), and can be sensitive to the assumptions made.

Time complexity

- We will use time complexity to define "efficiently solvable".
- But setting a specific threshold (e.g., $3n^5$) is too arbitrary, and the meaning of "efficiently solvable" is then either tied to a specific type of Turing machine (computer) or can change with time (as technology improves).

Example time complexities

input size	time						
	n	n^2	n^3	n^4	…	2^n	10^n
10	10	100	1000	10000		1024	10^{10}
20	20	400	8000	160000		1048576	10^{20}
30	30	900	27000	810000		1073741824	10^{30}
40	40	1600	64000	2560000		1099511627776	10^{40}
…							
100	10^2	10^4	10^6	10^8		$\approx 10^{30}$	10^{100}
…							
1000	10^3	10^6	10^9	10^{12}		$\approx 10^{300}$	10^{1000}
…							
10000	10^4	10^8	10^{12}	10^{16}		$\approx 10^{3000}$	10^{10000}

If you …

	n^c time	exponential time
increase input size by a fixed *amount* … (i.e., $n \rightarrow n+k$)	… then time increases by an *amount* $O(n^{c-1})$	… then time increases by a fixed *factor*
increase input size by a fixed *factor* k … (i.e., $n \rightarrow kn$)	… then time increases by fixed *factor* k^c	… then time is raised to *power* k
double your computer's speed …	… then you increase feasible input size by some fixed *factor*.	… then you increase feasible input size by some fixed *amount*.
need to handle inputs which are twice as large as those you can solve now …	… then you must wait for c years before computers are fast enough. *	… then the number of years you must wait is proportional to your current input size. *

* assumes Moore's Law: processor speed doubles every two years.

Polynomial time

A Turing machine M has **polynomial time complexity** if its time complexity is $O(n^k)$, for some fixed k.

$$t_M(n) = O(n^k).$$

- The power, k, is fixed.

  It does not depend on the input.

  But different polynomial time Turing machines often have different k's.

The class P

A language is **polynomial time decidable** if it can be decided by a polynomial time Turing machine.

The class of all languages decidable in polynomial time is called **P**

(which stands for **P**olynomial time).

This is the simplest (and, historically, the first) formal notion of "*efficiently solvable*".

The class P: some members

- {strings that end in b}
  - time complexity = $n + 1 = O(n)$ ✔
- {palindromes}
  - time complexity = $\frac{1}{2} . n(n+2) = O(n^2)$ ✔
- {the empty string}
  - time complexity = $O(1) = O(n^0)$ ✔
- any regular language
  - time complexity = … ✔
- any context-free language
  - time complexity = … ✔

The class P: properties

- P has been defined using a particular type of Turing machine M

  (two symbols, single one-way-infinite tape, moves one step left or right, …)

- But what if we used a different type of machine?

  Or some other model of computation?

  (E.g., your laptop, a smart phone, CSIRAC, TaihuLight, Summit, …)

- Wouldn't we get a different class P ?

The class P: properties

Suppose that

- computer M_1 has time complexity $t(n)$.
- computer M_2 can simulate machine M_1.

  (So M_2 is universal, e.g., a UTM.)
- any computation that takes time t on M_1 takes time $\leq c\, t^k$ on M_2.

(polynomial slowdown)

Then: if M_1 is polynomial time then M_2 takes polynomial time to simulate M_1 on input strings for M_1.

The class P: properties

Proof: If M_1 takes polynomial time, then $t(n) = O(n^K)$ for some fixed K. In other words, there exist c_1 and K such that $t(n) \leq c_1 n^K$ for sufficiently large n.

Time taken by M_2 to simulate M_1 on an input of size n is $\leq c\, t(n)^k$

$\leq c\, (c_1 n^K)^k$ for sufficiently large n

$\leq c\, c_1^k\, n^{Kk}$

$\leq c' n^{k'}$, where $c' = c\, c_1^k$ and $k' = K k$

(note, both constants)

$= O(n^{k'})$.

End of proof.

The class P: properties

It follows that, for such M_1 and M_2 :

If a language L can be decided in polynomial time using M_1, then it can be decided in polynomial time using M_2.

In fact, virtually any two computers can play the roles of M_1 and M_2 here. This is because any "reasonable" computer can simulate any other computer with at most polynomial slowdown.

The class P: properties

So the class P is independent of the particular model of computation used to define it.

This is reminiscent of the history of decidability: different paths can be taken to formulate the definition, using different models of computation, but it turns out that they all lead to the same class of decidable languages.

History of P :

Alan Cobham (1965), Jack Edmonds (1965), Michael Rabin (1966), …

The class P: more members

- the set of pairs of strings in lexicographic order
- the set of strings of matching parentheses
- the set of pairs of numbers that are coprime (a.k.a. relatively prime) [Euclidean algorithm]
- the set of square numbers
- the set of prime numbers [only proved in 2002] (Agrawal, Kayal, Saxena, *Annals of Mathematics*, 2004)
- the set of invertible matrices [MAT1841/MAT2003]
- the set of trees
- the set of balanced binary trees

The class P: more members

- the set of connected graphs
- {(G,s,t,k) : G has an s-t path of length ≤ k }
- the set of regular graphs (i.e., all vertices have the same degree)
- the set of 2-colourable graphs

  (graphs whose vertices can each be coloured Black or White, so that adjacent vertices receive different colours)
- the set of Eulerian graphs [MAT1830/FIT1029]
- the set of planar graphs

  (i.e., graphs which can be drawn in the plane so that no two edges cross, except possibly at their endpoints) [advanced]

The class P: more members

- 2-SAT: the set of satisfiable Boolean expressions in Conjunctive Normal Form (CNF), with two literals per clause
  - Boolean **variable** (x, y, …): takes value True or False
  - **Literal**: either a variable, or its logical negation.
    - If x is a variable then its literals are x and ¬x.
  - **Clause**: a disjunction of literals (e.g., $x \vee \neg y \vee \neg z$)
  - **CNF**: a conjunction of clauses.
    - E.g.: $(\neg x \vee \neg y) \wedge (x \vee \neg y \vee \neg z) \wedge (z) \wedge (y \vee \neg y \vee z)$
  - The "**2-**" in the name indicates that every clause has exactly two literals.
    - E.g.: $(\neg x \vee \neg y) \wedge (x \vee \neg z) \wedge (y \vee z) \wedge (y \vee \neg y)$

The class P: more members

- 2-SAT (continued):
  - A **truth assignment** is an assignment of a truth value (True/False) to each of the variables.

    I.e., a function $f : \{\text{variables}\} \rightarrow \{\text{True, False}\}$

    - E.g., f(x) = False, f(y) = False, f(z) = True

      is a truth assignment for the expression

      $$(\neg x \vee \neg y) \wedge (x \vee \neg z) \wedge (y \vee z) \wedge (y \vee \neg y),$$

      and it gives the expression the value

      $$(\neg F \vee \neg F) \wedge (F \vee \neg T) \wedge (F \vee T) \wedge (F \vee \neg F)$$

      $$= \ \ldots \ = \ \text{False}$$

    - Do *all* truth assignments make this expression False?

The class P: more members

- 2-SAT (continued):
  - An expression is **satisfiable** if it has a truth assignment which makes the expression True.
    - E.g., the expression

      $$(\neg x \vee \neg y) \wedge (x \vee \neg z) \wedge (y \vee z) \wedge (y \vee \neg y)$$

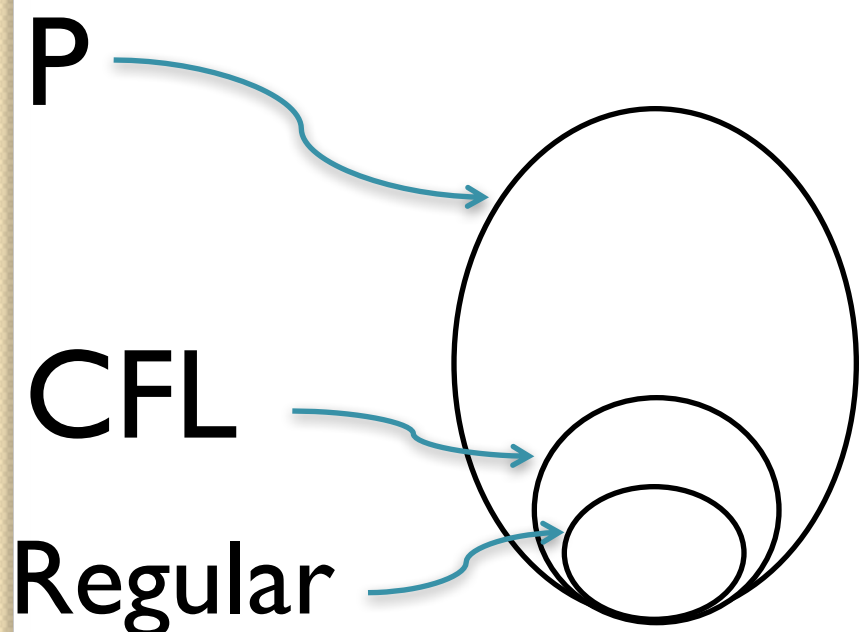
      is satisfiable, since the truth assignment

      $g(x) = \text{True},\ g(y) = \text{False},\ g(z) = \text{True}$

      makes the expression True.

  - Challenge: show that 2-SAT is in P.

P and other language classes



In P, not in CFL:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
$\{ a^n b^n c^n : n \geq 1 \}$,
…

In CFL, not Regular:

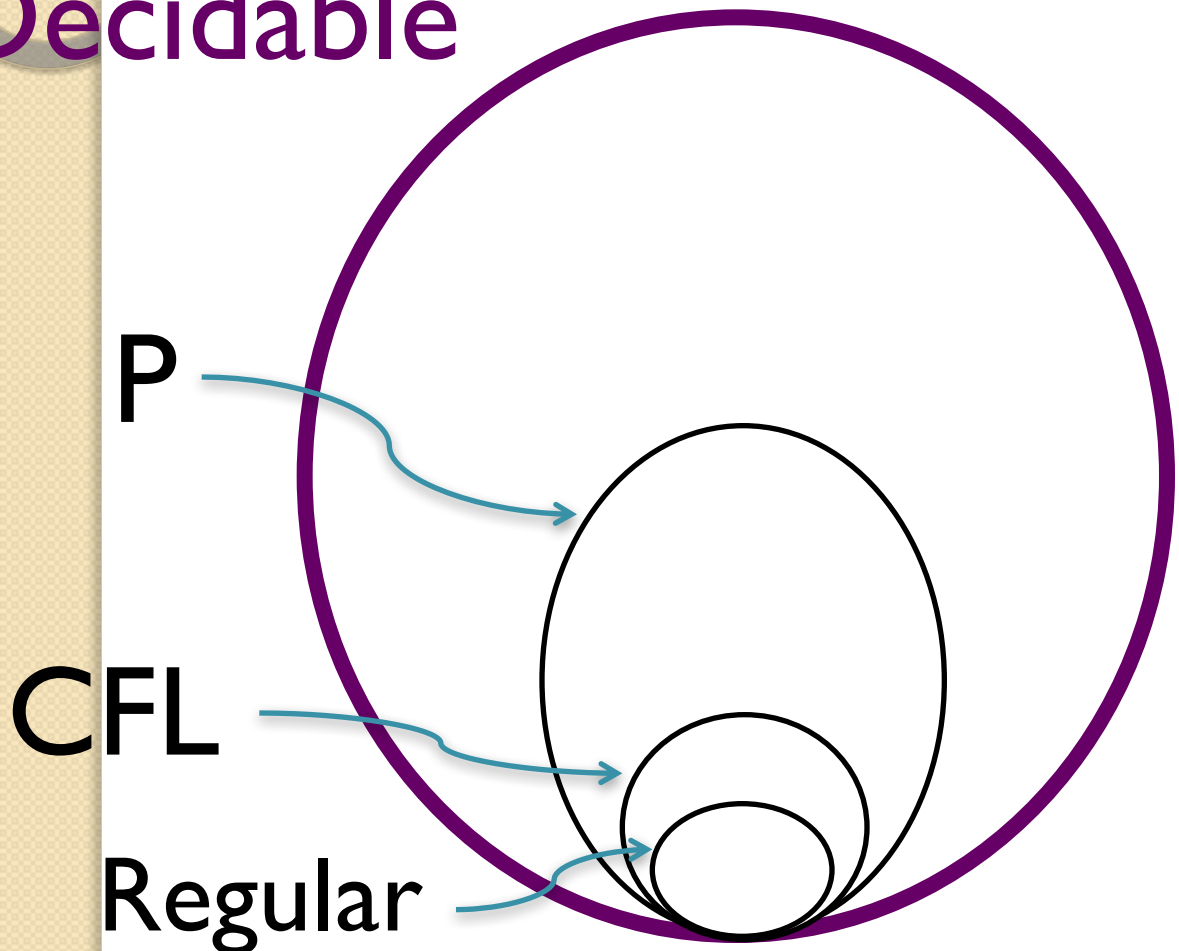
$\{ a^n b^n : n \geq 1 \}$,
PALINDROMES,
DYCK, …

Regular:

EVEN-EVEN,
Multiples of 3,
CWL, …

P and other language classes

Decidable



Decidable, not in P:

Generalised Chess,
Generalised Go,
Equivalence for regexps with exponentiation,
…

Revision

- Time complexity
- Practical decidability
- Definition of P
- Relationship between P and model of computation (and argument using simulation and polynomial slowdown)
- Languages in P

- Reading: Sipser, sections 7.1-7.2