Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 24
Polynomial time, and the class P

slides by Graham Farr

Overview

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

Decidable languages

Decidable languages are "solvable", in principle, by a computer.

It 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

For any Turing machine M, define:

$$\begin{aligned} t_M(x) \quad &:= \quad \text{time taken by } M \text{ for input } x \\ &:= \quad \#\text{ steps } M \text{ takes until it halts.} \end{aligned}$$

Time complexity:

$$\begin{aligned} t_M(n) \quad &:= \quad \text{maximum time taken by } M \text{ for any input of length } n \\ &= \quad \max \{ t_M(x) : |x| = n \}. \end{aligned}$$

It's a function of n, which can be any positive integer.

For it to be defined, M must halt on all inputs, i.e., M must be a decider.

It's a *worst case* measure.

Time complexity

Examples:

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

$$\text{Finite Automaton} \quad \longrightarrow \quad \text{Turing machine}$$

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 FIT1045 or FIT1008, you may have 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	n^2	n^3	n^4	$\cdots$	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}
$\vdots$					$\cdots$		
100	10^2	10^4	10^6	10^8		$\approx 10^{30}$	10^{100}
$\vdots$							
1000	10^3	10^6	10^9	10^{12}		$\approx 10^{300}$	10^{1000}
$\vdots$							
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., $n \longrightarrow n + k$)	... 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., $n \longrightarrow kn$)	... 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 $2c$ years before computers are fast enough. *	... then your 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.

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

This means that there is a constant c such that, for all *sufficiently large* n,

$$t_M(n) \leq c \cdot n^k.$$

Symbolically:

$$\exists c \, \exists N \, \forall n \; : \; n > N \implies t_M(n) \leq c \cdot n^k.$$

The power, k, is fixed.
It does not depend on the input.

But different polynomial time Turing machines often have different k.

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 Polynomial 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	$=$	$\frac{1}{2} \cdot (n+1)(n+2)$	$=$	$O(n^2)$	✓
$\{\varepsilon\}$	time complexity	$=$	$O(1)$	$=$	$O(n^0)$	✓
any regular language	time complexity	$=$	$\ldots$			✓
any context-free language	time complexity	$=$	$\ldots$			✓

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 smartphone, CSIRAC, TaihuLight, LUMI, Fugaku, Frontier, ...)

Would 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.
  (E.g., M_2 is a UTM.)
- any computation that takes time t on M_1 takes time $\leq ct^k$ 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 ...

$$
\begin{aligned}
&\leq c \cdot t(n)^k \\
&\leq c\,(c_1 n^K)^k \quad \text{for sufficiently large } n \\
&\leq c\, c_1^k\, n^{Kk} \\
&\leq c'\, n^{k'}, \qquad \text{where } c' = cc_1^k \text{ and } k' = Kk \quad \text{(note, both constants)} \\
&= O(n^{k'}).
\end{aligned}
$$

$\square$

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 (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:

$$\{(G, s, t, k) : G \text{ has an } s\text{–}t \text{ path of length} \leq k\}$$

- 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/FIT1045]
- 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

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 to each of the variables, i.e., a function $f : \{\,\text{variables}\,\} \to \{\,\text{True}, \text{False}\,\}$

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

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

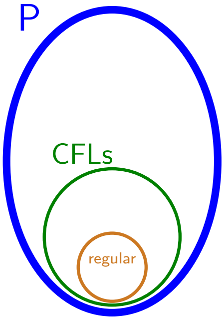
is satisfiable, since the truth assignment

$$\begin{aligned} g(x) &= \text{True} \\ g(y) &= \text{False} \\ g(z) &= \text{True} \end{aligned}$$

makes the expression True.

Challenge: show that 2-SAT is in P.

P and other language classes



In P, not in CFL:

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

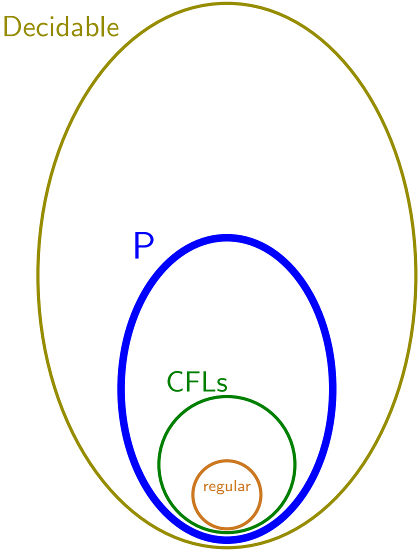
In CFL, not Regular:

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

Regular:

EVEN-EVEN, Multiples of 3, CWL, ...

P and other language classes



Decidable, not in P:

Time-bounded Halting Problem, Generalised Go, Regexp Equivalence, Generalised Chess, ...

In P, not in CFL:

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

In CFL, not Regular:

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

Regular:

EVEN-EVEN, Multiples of 3, CWL, ...

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
- M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco, 1979, Chapter 1 and especially the first two sections of Chapter 2: §2.1, §2.2.