Complexity: moving from qualitative to quantitative considerations

Textbook chapter 7

Complexity Theory: study of what is computationally feasible (or tractable) with limited resources:

- **running time** (main focus)
- storage space
- number of random bits
- degree of parallelism
- rounds of interaction
- others...

more on some of those in complexity (HT)

"polynomial-time reduction" can prove computational hardness results, analogous to reductions/undecidability

Worst-case analysis

Always measure resource (e.g. running time) in the following way:

- as a function of the input length
- value of the function is the maximum quantity of resource used over all inputs of given length
- called "worst-case" analysis

"input length" is the length of input string

Note: the question of how run-time scales with input size, is unaffected by the speed of your computer

Time complexity

Given language L recognised by some TM M, we can use number of steps of M as precise notion of computational runtime.

But this function shouldn't be studied in too much detail—

- We do not care about fine distinctions
  - e.g. how many additional steps M takes to check that it is at the left of tape
- We care about the behaviour on large inputs
  - general-purpose algorithm should be "scalable"
  - overhead for e.g. initialisation shouldn't matter in big picture

Time complexity

- Measure time complexity using asymptotic notation ("big-oh" notation)
  - disregard lower-order terms in running time
  - disregard coefficient on highest order term
- example

$$f(n) = 6n^3 + 2n^2 + 100n + 102781$$

  - "f(n) is order n^3"
  - write $f(n) = O(n^3)$

E.g. We might consider the class of "Cubic time decision problems" ($O(n^3)$) problems
We will never consider the class of "$2n^3 + 17$ time decision problems"
in practice, usually the constant hidden by big-oh notation isn't super-important...

big-oh notation: textbook chapter 7.1

Asymptotic notation

Definition

given functions $f, g : \mathbf{N} \to \mathbf{R}^+$, we say $f(n) = O(g(n))$ if there exist positive integers c, n_0 such that for all $n \geq n_0$

$$f(n) \leq cg(n)$$

- meaning: $f(n)$ is (asymptotically) less than or equal to $g(n)$
- if g always > 0 can assume $n_0 = 0$, by setting

$$c' = \max_{0 \leq n \leq n_0} \{c, f(n)/g(n)\}$$

Time complexity of language 0^k1^k

On input x:

- scan tape left-to-right, reject if 0 to right of 1
- repeat while 0's, 1's on tape:
  - scan, crossing off one 0, one 1
- if only 0's or only 1's remain, reject; if neither 0's not 1's remain, accept

$O(n)$ steps

$\leq n$ repeats
$O(n)$ steps

$O(n)$ steps

total $= O(n) + n.O(n) + O(n) = O(n^2)$

Important "Big O" Classes

- "logarithmic": $O(\log n)$
  - $\log_b(n) = (\log_2 n)/(\log_2 b)$
  - so $\log_b(n) = O(\log_2(n))$ for any constant b; therefore suppress base when we write it
- "polynomial": $O(n^c) = n^{O(1)}$
- "exponential": $O(2^{n^\delta})$ for $\delta > 0$

Time complexity classes

Recall:

- language is a set of strings
- a complexity class is a set of languages
- complexity classes we've seen:
  - Regular languages, Context-free languages, Decidable languages, CE Languages, co-CE languages

Definition

$TIME(t(n)) = \{L :$ there exists a TM M that decides L in time $O(t(n))\}$

A priori, $TIME(t(n))$ could be a different class for every function t

Robustness of Complexity

At this point we could begin to draw pictures of the relationship of time classes (e.g. $TIME(n^3)$, $TIME(2^n)$,...) to other classes we know of.

But before we do, ask: how "robust" are these classes?

- Do the precise details of the variation of TM we use matter (e.g. single-tape vs. multi-tape, one head move per transition vs. several, acceptance by state only vs. ...)?
- Could we use C or Java instead of TMs in defining "time steps"? Does it matter if we use C vs. FORTRAN in this?

Robust complexity

- Complexity of $L = \{0^k 1^k : k \geq 0\}$
- On a Turing Machine it is easy to do in $TIME\ O(n^2)$.
- Book: it is also in $TIME(n \log n)$ by giving a more clever algorithm
- Can prove: $O(n \log n)$ time required on a single tape TM.
- How about on a multitape TM?

2-tape TM M deciding $L = \{0^k 1^k : k \geq 0\}$.

On input x:

- scan tape left-to-right, reject if 0 to right of 1 — $O(n)$
- scan 0's on tape 1, copying them to tape 2 — $O(n)$
- scan 1's on tape 1, crossing off 0's on tape 2 — $O(n)$
- if all 0's crossed off before done with 1's, reject
- if 0's remain after done with ones, reject; otherwise accept

total:
$3 * O(n) = O(n)$

Multitape TMs

Convenient to "program" multitape TMs rather than single ones

- equivalent when talking about decidability
- not equivalent when talking about time complexity

The speed-up of using multi-tape machine turns out to be only quadratic:

Theorem

Let $t(n)$ satisfy $t(n) \geq n$. Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$.

Textbook, Theorem 7.8

Multitape TMs

- Moral 1: feel free to use k-tape TMs, but be aware of slowdown in conversion to TM
- Moral 2: $O(n)$ is not super-robust. Polynomial time ($TIME(n^c)$ for some c) and exponential time (2^{n^c} for some c) are more stable under tweaking machine model. High-level operations you are used to using can be simulated by TM with only polynomial slowdown

  e.g., copying, moving, incrementing/decrementing, arithmetic operations $+$, -, *, /

We will focus on these coarse-but-robust classes.

A Robust Time Complexity Class

interested in a coarse classification of problems. For this purpose,

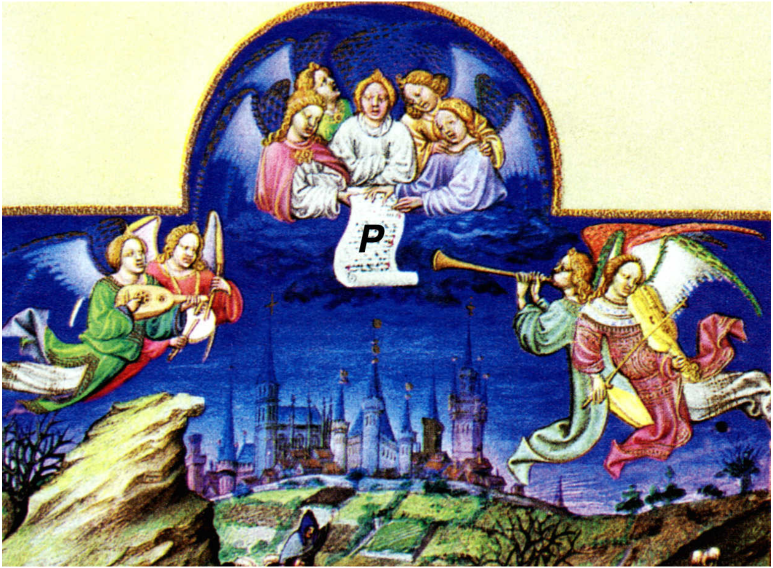
- treat any polynomial running time as "efficient" or "tractable"
- treat any exponential running time as inefficient or "intractable"

Key definition:
"**P**" or "polynomial-time" or PTIME

$$\mathbf{P} = \cup_{k \geq 1} TIME(n^k)$$

"Think of **P** as standing for Practical" —Tim Gowers

The complexity class **P**



Positive results: Examples of languages in **P**

Most "school algorithms" are easily seen to be in **P**.

- Standard arithmetic operations ($\times, +$ etc) on (e.g.) binary numbers.
- Searching for an item in a list.
- Sorting

Can use "robustness" of **P** in proving positive results.

Language Map Revisited

Key definition

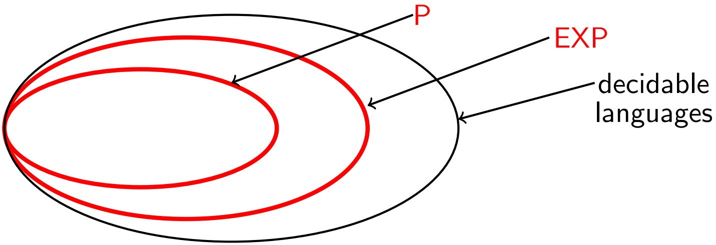
"**P**" or "polynomial-time" or PTIME

$$\mathbf{P} = \cup_{k \geq 1} TIME(n^k)$$

Definition

"**EXP**" or "exponential-time" or EXPTIME

$$\mathbf{EXP} = \cup_{k \geq 1} TIME(2^{n^k})$$



$$\mathbf{P} \subseteq \mathbf{EXP} \subseteq 2^{\mathbf{EXP}} \subseteq \cdots$$

(similar to undecidability of HALT:)

- TM[i,j] = Acts like ith Turing Machine M_i but reject w if no acceptance after $j \cdot |w|^j + j$ steps
  - Languages accepted by TM[i,j]'s are exactly the polynomial time languages
  - Diagonal machine $Diag_P$: on input $w = a^i b^j$ run TM[i,j] on w and then do the opposite
- How fast is $Diag_P$?

So, artificial language outside **P**, in EXPTIME

Related:

- $\{\langle M, j, k, w\rangle : TM[j,k] \text{ accepts } w\}$

In the book you can find a similar example:
$ACC_{Bounded} = \{\langle M, w, j\rangle : M$ is a TM, j binary representation of an integer, M accepts w within at most j steps$\}$
i.e. roughly
$\{\langle M', w\rangle : M'$ is a PTIME machine and M' accepts $w\}$

Time Hierarchy Theorem

Theorem

For every proper complexity function $f(n) \geq n$*:*

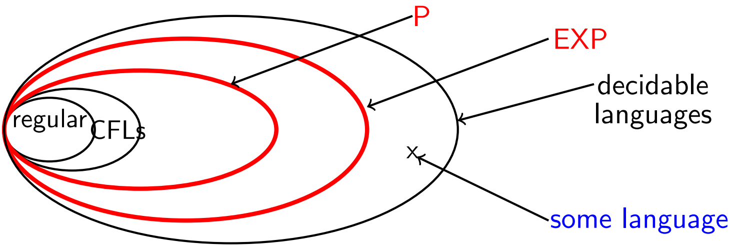
$$TIME(f(n)) \subsetneq TIME((f(2n))^3).$$

Most natural functions (and 2^n in particular) are proper complexity functions. We will ignore this detail in this class. We do not cover the proof in this course. But understand the conclusions: $TIME(n) \subsetneq TIME(n^3)$ and $TIME(2^{(n/6)}) \subsetneq TIME(2^n)$, etc.

This tells us that **P** differs from **EXP**

Bootstrapping from examples

We have defined the complexity classes **P** (polynomial time), **EXP** (exponential time)

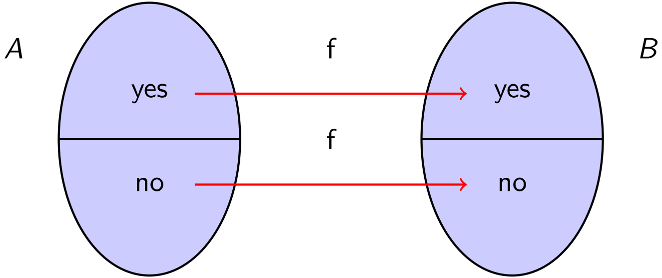


How do you bootstrap to show something is not in **P**, not in **EXP**, etc.?

Poly-time reductions

Type of reduction we will use:

- "many-one" poly-time reduction (commonly)
- "mapping" poly-time reduction (book)



reduction from language A to language B

Poly-time reductions

Definition

$A \leq_P B$ ("A reduces to B") if there is a poly-time computable function f such that for all w

$$w \in A \Leftrightarrow f(w) \in B$$

- as before, condition equivalent to:
  YES maps to YES *and* NO maps to NO
- as before, meaning is:
  B is at least as "hard" (or expressive) as A

Poly-time reductions

Theorem

If $A \leq_P B$ *and* $B \in \mathbf{P}$ *then* $A \in \mathbf{P}$.

Proof.

A poly-time algorithm for deciding A:

- on input w, compute $f(w)$ in poly-time.
- run poly-time algorithm to decide if $f(w) inB$
- if it says "yes", output "yes"
- if it says "no", output "no"

□

In particular, once you know some concrete language L is not in **P** (**EXP**, etc.), you can use reductions to show that other languages are not in **P**.

The way you show something is in **P**:
Give a PTIME algorithm
Also can do via reductions.

The way you show something is not in **P**:
Reduce from problem known not to be in **P**
(e.g. acceptance problems)

Problem: REACH=Given a graph, and two nodes n_1 and n_2, decide if there is a path from n_1 to n_2.

In **P** Dynamic programming

Problem: HAM=Given a graph G, find out if there is a circuit that hits every node exactly once.

(stands for Hamiltonian Circuit).
Obvious algorithm shows that is in exponential time.

Is it in **P**? Unknown!

Can we show HAM is not in **P**?

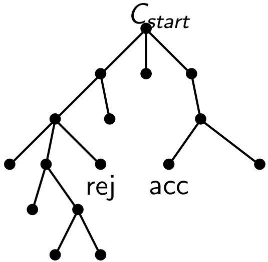
- Don't know how to. Believed unlikely to be in PTIME. But probably cannot reduce from a known EXPTIME problem
- Why is it difficult to show HAM is not in **P**? There is an important positive feature of HAM that makes it "close to PTIME"

HAM is decidable in polynomial time by a nondeterministic TM

Nondeterministic TMs

- informally, TM with several possible next configurations at each step
- formally, an NTM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where: everything is the same as a TM except the transition function: $\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$

visualize computation of a NTM M as a tree



- nodes are configurations
- leaves are accept/reject configurations
- M accepts if and only if there exists an accept leaf
- We are interested in NTMs where no paths go on forever:
- allows us to define running time on string w as: length of longest path (depth of tree)

The class **NP**

Recall Definition: $TIME(t(n)) = \{L$: there exists a TM M that decides L in time $O(t(n))\}$

$$\mathbf{P} = \cup_{k \geq 1} TIME(n^k)$$

New Definition: $NTIME(t(n)) = \{L$: there exists a NTM M that decides L in time $O(t(n))\}$

$$\mathbf{NP} = \cup_{k \geq 1} NTIME(n^k)$$

NP Languages

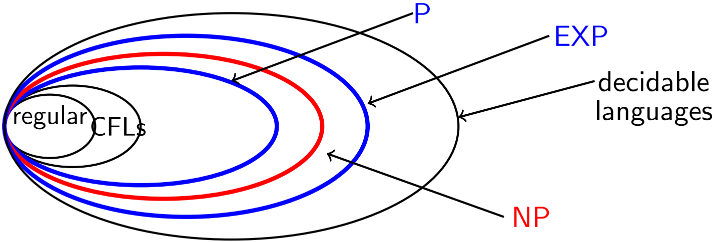
Informally: Languages L where membership can be done through run making polynomial-sized "guesses", and then verifying a guess in polynomial time.

Need to know:

- Every run-with-guesses is polynomial sized
- If the input is in L, some guess will succeed.
- If the input is not in L, no guess will succeed.
- Can verify that a guess is correct.

NP: computational challenges where solutions are easy to check, but may be hard to find

NP in relation to P and EXP



- **P** $\subseteq$ **NP** (poly-time TM *is* is poly-time NTM
- **NP** $\subseteq$ **EXP**
  - configuration tree of n^k-time NTM has $\leq b^{n^k}$ nodes, where b is max number of choices per state
  - can traverse entire tree in $O(b^{n^k})$ time

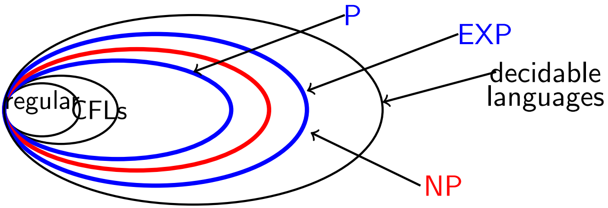
we do not know if either inclusion is proper

NP

NTM=TM with several next configurations at each step

formally, an NTM is a 7-tuple
$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where: everything is the same as a TM except the transition function:
$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$

NP Machine – maximum length of runs on w is polynomially bounded in $|w|$



NP problems

...include all problems known to be in **P**, then e.g.

Travelling Salesman Problem

Given an *edge weighted graph* G – edges have weights (distances) – and an integer k, does G have a Hamiltonian circuit with sum of weights below k?

Given a graph G does G have a 3-colouring (labelling of nodes with 3 colours such that no two adjacent nodes have the same colour)?

Given a graph G and a number k (in binary), does G have a clique of size k?

Are these problems in **P**?

More general open question (for 40 years): does $\mathbf{P} = \mathbf{NP}$?

Clay Institute Prize for solving this: $1 million

What can we show about **NP**-problems?

Weaker thing than showing a problem is not in **P**

Show if problem is in **P***, then every* **NP** *problem is in* **P**

This means: problem is as hard as any **NP** problem i.e. as hard as TSP, as hard as ...

Hardness and completeness

Recall:

- a language L is a set of strings
- a complexity class C is a set of languages

Definition

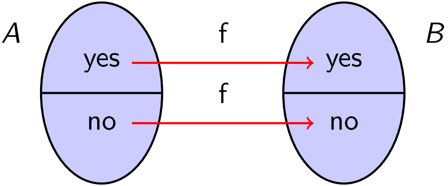
a language L is C-hard (under polynomial time reductions) if for every language $A \in C$, A poly-time reduces to L; i.e., $A \leq_P L$.

meaning: L is at least as "hard" as anything in C

NP-hard – every **NP** language reduces to it

Hard Problems, in General

Polynomial-time reductions



Hard problem for class C: language L such that every other problem in C reduces in polytime to L

E.g. L is **EXPTIME**-hard: every **EXPTIME** problem reduces to L (hence L is not in **P**, since **P**$\neq$EXPTIME)

L is **NP**-hard: every **NP** problem reduces to L

Complete problem for class C: Language that is in **C** and is **C**-hard

L is **NP**-complete: $L \in$ **NP** and every **NP** problem reduces to L

= "Hardest problem in **NP**"

P=**NP**??

Are TSP, 3 Coloring, Clique and other NP problems in **P**?
Open question for 40 years – does **P**=**NP**?
We do not know how to show that **NP** problems are not in **P**
We do know how to show that problems are **NP**-hard
If a problem *L* is shown to be **NP**-hard, this means:
If we can show *L* is in **P**, we will be rich and famous
It will be extremely difficult to find a PTIME algorithm for *L*

General belief in complexity community: it is extremely unlikely that there is a PTIME algorithm for *L*
NP-hard problems often called "presumably intractable"

An artificial **NP**-complete problem

recall: "C-complete" means, "in C, and at least as hard as anything in C"

Version of ACC_{TM} with a unary time bound, and NTM instead of TM:

$ANTM_U = \{\langle M, x, 1^m \rangle : M$ is a NTM that accepts x within at most m steps$\}$

Theorem

*$ANTM_U$ is **NP**-complete.*

Proof:

An artificial **NP**-complete problem

recall: "C-complete" means, "in C, and at least as hard as anything in C"

Version of ACC_{TM} with a unary time bound, and NTM instead of TM:

$$ANTM_U = \{\langle M, x, 1^m \rangle : M \text{ is a NTM that accepts } x \text{ within at most } m \text{ steps}\}$$

Theorem

$ANTM_U$ is **NP***-complete.*

Proof:

Part 1. Need to show $ANTM_U \in \mathbf{NP}$.

- simulate NTM M on x for m steps; do what M does
- $n =$ length of input $\langle M, x, 1^m \rangle \geq m$
- running time is some constant factor of $|x| + (|M|^* m) \leq n^2$

An artificial **NP**-complete problem

$ANTM_U = \{\langle M, x, 1^m \rangle : M$ is a NTM that accepts x within at most m steps$\}$

Proof that $ANTM_U$ is **NP**-hard:

- Given **NP** problem A, must poly-reduce to $ANTM_U$
- TM M_A for A has time bound $t(|w|) = O(|w|^k)$ for some k
  Define: $f(w) = \langle M_A, w, 1^m \rangle$ where $m = t(|w|)$
- is $f(w)$ poly-time computable?
  - hardcode M_A and k...
- YES maps to YES?
  - $w \in A \Rightarrow \langle M_A, w, 1^m \rangle \in ANTM_U$
- NO maps to NO?
  - $w \notin A \Rightarrow \langle M_A, w, 1^m \rangle \notin ANTM_U$

An artificial **NP**-complete problem

Conclude: If you can find a poly-time algorithm for $ANTM_U$ then there is automatically a poly-time algorithm for every problem in **NP** (i.e., **NP**=**P**).

Want to know if natural problems (e.g. TSP, HAM, etc.) are **NP**-hard.

Start with one natural problem, involving propositional logic. From there go to graph problems.

Propositional Logic

A *propositional variable* takes value either TRUE or FALSE.

A *propositional formula* is built up from propositional variables and the constants TRUE or FALSE, using operators (or "connectives") like AND ($\wedge$), OR ($\vee$), NOT ($\neg$), IMPLIES ($\Rightarrow$), etc.
Suppose $x_1, x_2, \ldots$ are propositional variables. Example formula:

$$(x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_4) \vee (x_1 \Rightarrow x_5)$$

By the way, technically all boolean operations can be expressed in terms of NAND, but it's useful to use at least $\wedge$, $\vee$, $\neg$.

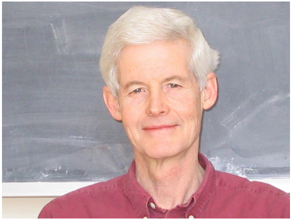
Some more jargon: an assignment of truth values to the variables is sometimes called a "world"; a formula ϕ that always evaluates to TRUE (for any world) is a tautology (or valid), if ϕ is always FALSE it's a contradiction.
Usually I don't say "world", I say "truth assignment", or "(non)-satisfying assignment" (w.r.t. some formula)

2 problems involving propositional logic

1. Given a formula ϕ on variables $x_1, \ldots x_n$, and values for those variables, derive the value of ϕ — easy!
2. Search for values for $x_1, \ldots, x_n$ that make ϕ evaluate to TRUE — naive algorithm is exponential: 2^n vectors of truth assignments.

Cook's Theorem (1971):

The second of these, called SAT, is **NP**-complete.



Stephen Cook

The challenge of solving boolean formulae

There's a HUGE theory literature on the computational challenge of solving various classes of syntactically restricted classes of boolean formulae, also circuits.

Likewise much has been written about their relative *expressive power*

SAT-solver: software that solves input instances of SAT — OK, so it's worst-case exponential, but aim to solve instances that arise in practice. Need smart algorithms (not truth table!)

Reducing an **NP** problem to SAT

Goal: fixing non-deterministic TM M, integer k, given w create in poly-time a propositional formula $\mathbf{CodesAcceptRun}_M(w)$ that is satisfied by assignments that code an n^k length accepting run of M on w (where $n = |w|$)

The propositional variables "describe" an accepting computation, e.g. $HasSymbol_{i,j}(a)$ is TRUE if the computation has symbol a on the j-th tape position at step i.

We'll assume M has "stay put" transitions for which it can change tape contents; R and L moves don't change tape. Assume also that to accept, M goes to LHS of tape and prints special symbol.

Reducing an **NP** problem to SAT

Goal: fixing non-deterministic TM M, integer k, given w create in poly-time a propositional formula $\mathbf{CodesAcceptRun}_M(w)$ that is satisfied by assignments that code an n^k length accepting run of M on w (where $n = |w|$)

Tape space j

		1	2	$\cdots$	n^k
Time i	1	(q_0, w_1)	w_2	$\cdots$	
	2	w'_1	(q_1, w_2)		
	$\vdots$				
	$\vdots$				
	n^k				

This corresponds to a run where
$HasSymbol_{1,1}(w_1)$
$HasHead_{1,1}(q_0)$
$HasSymbol_{1,2}(w_2)$
$HasSymbol_{2,1}(w'_1)$
$HasSymbol_{2,2}(w_2)$
$HasHead_{2,2}(q_1)$
...are true
(Others, e.g. $HasHead_{1,2}(q_0)$ are false)

Idea: the search for "correct" non-determinstic choices for M shall correspond to search for satisfying assignment for $\mathbf{CodesAcceptRun}_M(w)$.
$\mathbf{CodesAcceptRun}_M(w)$ shall be a conjunction of *clauses*.

Moving head clauses: leftward-moving State

Leftward moving state. If M has transition rule $(q,a) \rightarrow \{(q_1,a,L),(q_2,a,L)\}$ then we write:

$$HasHead_{i,j}(q) \Rightarrow [HasHead_{i+1,j-1}(q_1) \vee HasHead_{i+1,j-1}(q_2)]$$

Write the above for all $i,j \in \{1,2,3,\ldots,n^k\}$.

Tape space

Time

	1	$\cdots$	$j-1$	j	$\cdots$	n^k
1						
i			w_2	(q,a)		
$i+1$			(q_1,w_2)	a		
$\vdots$						
n^k						

Moving head clauses: Rightward-moving State or Leftward-moving State

For every rightward or leftward state q, for every a we add the clause:

$$HasSymbol_{i,j}(a) \wedge HasHead_{i,j}(q) \Rightarrow HasSymbol_{i+1,j}(a)$$

Meaning: if the head is at place j at step i and we are in a rightward- or leftward moving state, symbol in place j at step $i+1$ is the same.

Tape space

	1	$\cdots$	j		$\cdots$	n^k
1						
Time i			(q, a)	w_2	$\cdots$	
$i+1$			a	(q_1, w_2)	$\cdots$	
$\vdots$						
n^k						

Moving head clauses: stay-same state

For every stay-and-write state q, if we have transition $(q, w_0) \rightarrow \{(q_1, w_1, Stay), (q_2, w_1, Stay)\}$ then we add:

$$HasSymbol_{i,j}(w_0) \wedge HasHead_{i,j}(q) \Rightarrow HasSymbol_{i+1,j}(w_1)$$

(new symbol is written – use "stay determinism" assumption of M_A here!) And also:

$$HasHead_{i,j}(q) \Rightarrow [HasHead_{i+1,j}(q_1) \vee HasHead_{i+1,j}(q_2)]$$

(head does not move, although state may change)

	1	$\cdots$	j		$\cdots$	n^k
1						
i			(q, w_0)	$\cdots$		
$i+1$			(q_1, w_1)	$\cdots$		
$\vdots$						
n^k						

TM head “sanity clauses”

Include the following:

$$\mathit{HasHead}_{i,j}(q) \Rightarrow \neg \mathit{HasHead}_{i,j'}(q')$$

...for all states q, q', for all i, j, j' with $j \neq j'$.

More sub-formulae for Transitions: away from head clauses

Clauses stating that if the head is not close to place j at time i, then symbol in place j is unchanged in the next time.
For any state q and symbol w_3, any $i \leq n_k$ and number h in a certain range we have

$$HasHead_{i,j}(q) \wedge HasSymbol_{i,j+h}(w_3) \Rightarrow HasSymbol_{i+1,j+h}(w_3)$$

If q is a rightward-moving state, do this for $n^k - j \geq h \geq 2$ **and** $-(j-1) \leq h < 0$
If q is a leftward-moving state do this for $n^k - j \geq h \geq 1$ **and** $-(j-1) \leq h < -1$

If q is a stay put state, do this for $h \neq 0$

	1	$\cdots$	j	$\cdots$	n^k
1					
i			(q, w_0)	$\cdots$ w_3	
$i+1$			(q_1, w_1)	$\cdots$ w_3	
$\vdots$					

Final configuration clause: let's assume that whenever M accepts, it accepts at LHS of tape and prints special symbol $\square$ there

$$HasSymbol_{n^k,1}(\square) \wedge HasHead_{n^k,1}(q_{accept})$$

At time n^k, head is at the beginning and state is accepting with special termination symbol

	1	$\cdots$			$\cdots$	n^k
1	q_0	w_1	w_2	$\cdots$		
$\vdots$						
n^k	$(q_{accept}, \square)$					

Proof of the construction (that it's a poly-time reduction)

Recall: we had an arbitrary NTM M and running time bound.
Need to show $L(M) \leq_P SAT$, via f:words$\rightarrow$formulae
Let $Form_M(w)$ be result of f evaluated on w.

1. Show: $Form_M(w)$ is computable from w in PTIME

2. Show: If w is accepted by M, then $Form_M(w)$ is satisfiable

Take a run r witnessing acceptance of w, and let $Code(r)$ be the corresponding assignment. Verify that $Code(r)$ satisfies $Form_M(w)$: for each subformula in the conjunction, show that it follows from the properties of an accepting run.

3. Show: if $Form_M(w)$ is satisfiable, then w is accepted by M

Tougher direction – Take a satisfying assignment A of $Form_M(w)$. First show some sanity properties of A which indicate that it corresponds to a run.

Proof of the construction

Proving: If $Form_M(w)$ is satisfiable, then w is accepted by M

Take a satisfying assignment A of $Form_M(w)$. Want to show that there is an accepting run of M on w. First show some sanity properties of A which indicate that it corresponds to a run:

(a) For every $i < n^k$, there is some $j < n^k$ and q such that $HasHead_{i,j}(q)$ is true.
Prove by induction on i: for $i = 1$, follows from the initial state clause; induction step follows from the transition formulae.

(b) For each $i < n^k$, can't be 2 different $j < n^k$ and q with $HasHead_{i,j}(q)$ is true.
Follows from the "sanity clause".

Proof of the construction

Proving: If $Form_M(w)$ is satisfiable, then w is accepted by M

Take a satisfying assignment A of $Form_M(w)$. First show some sanity properties of A which indicate that it corresponds to a run:

(c) For every $i < n^k$, $j < n^k$ there must be some a such that $HasSymbol_{i,j}(a)$ holds

Prove the statement "for all j..." by induction on i.

$i = 1$ follows from the initial state clause; induction step follows from the head-moving clauses + "stay the same" clauses

Each of these formulae are of the form:

```
if (guards) then (Some Proposition holds at place i+1,j)
```

Argue, using induction, that one of the guard conditions has to hold at every j

(d) For every $i < n^k$, and $j < n^k$ can't be two different a such that $HasSymbol_{i,j}(a)$ holds

Follows from Row Sanity Clauses

Proof of the construction

Proving: If $Form_M(w)$ is satisfiable, then w is accepted by M

Take a satisfying assignment A of $Form_M(w)$. We have shown sanity properties of A which indicate that it corresponds to a run.
Now can **define** a sequence of configurations of M from A: config i has:

- tape value at place j of config i is the unique symbol a such that $HasSymbol_{i,j}(a)$ holds
- control state is the unique q such that $HasHead_{i,j}(q)$ holds for some j
- head is at the unique j such that $HasHead_{i,j}(q)$ for some q

Well-defined by (a)–(d). Show that this is an **accepting run** for w.

Verify each property of an accepting run.
Initial state ok?
→ follows from *initial state clause*
Transition function respected?
→ follows from *head-moving clauses* (for cells close to the head) and *away-from-head clauses* (for other cells)
Acceptance state reached at the end?
→ follows from *acceptance clause*

A propositional formula is in **Conjunctive Normal Form (CNF)** if it is of the form

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each C_i is of the form $(R_1 \vee \ldots \vee R_m)$ each R_i is either a proposition or its negation.

k-CNF means CNF where each C_i has $\leq k$ propositions.

3CNF example: $(p_1 \vee p_2) \wedge (\neg p_2 \vee p_3) \wedge (p_3 \vee p_4 \vee \neg p_5)$

Conjunction of Disjunctions Each of the C_i is called a **clause**

Checking whether a CNF is a validity is easy.

Checking whether a CNF is satisfiable is not so easy

returning to Cook's Theorem

Theorem

Checking whether a 3CNF propositional formula is satisfiable is **NP***-complete (***3-SAT** *is* **NP***-complete)*

Proof:

Previous argument produces a long conjunction of things of form:

$A \Rightarrow B$; can be rewritten $\neg A \vee B$

$A \wedge B \Rightarrow C$ can be rewritten $\neg(A \wedge B) \vee C = \neg A \vee \neg B \vee C$

$A \Rightarrow (B \vee C)$; can be rewritten $\neg A \vee B \vee C$

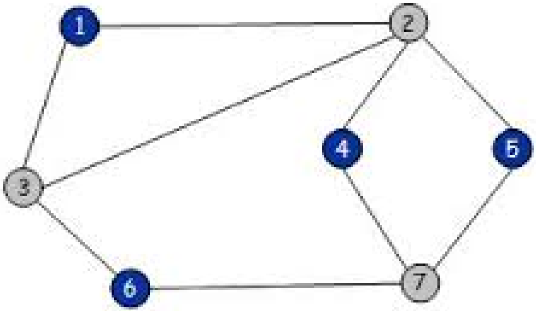
Powerful tool for negative results: prove that a problem L is **NP**-complete by reducing 3SAT to L

$$\text{3-SAT} \leq_P L \Rightarrow L \text{ is } \mathbf{NP}\text{-hard}$$

Show how this is done for a graph problem next

INDEPENDENT SET

Definition: given a graph $G = (V, E)$, an **independent set** in G is a subset $V' \subseteq V$ such that for all $u, w \in V'$, $(u, w) \notin E$.



INDEPENDENT SET is **NP**-complete

Theorem

the following language is **NP***-complete:*

$$\textbf{IS} = \{(G,k) : G \textbf{ has an independent set of size } \geq k\}.$$

Proof:

- Part 1: IS $\in$ **NP**. (Proof: exercise)
- Part 2: IS is **NP**-hard.
  - reduce from 3-SAT

INDEPENDENT SET is **NP**-complete

We are reducing **from the language:**

$$3\text{-SAT} = \{\phi : \phi \text{ is 3-CNF formula with a satisfying assignment}\}$$

to the language:

$$\text{IS} = \{(G, k) : G \text{ has an IS of size} \geq k\}.$$

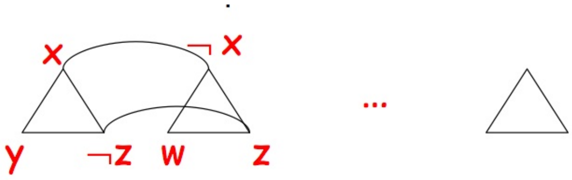
Given ϕ we must produce a G, k such that ϕ is satisfiable iff G has an IS of size $\geq k$.

INDEPENDENT SET is **NP**-complete

The reduction f: given

$$\phi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

we produce graph G_ϕ:



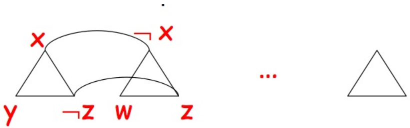
- **one triangle for each of m clauses — duplicate a literal[1] if it appears in multiple clauses**
- **additional edge between every pair of contradictory literals**
- **choose $k = m =$ number of clauses**

[1] A literal is a propositional variable or its negation

INDEPENDENT SET is **NP**-complete

$$\phi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

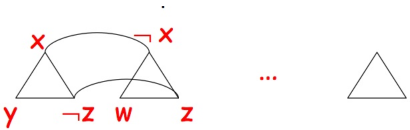
$f(\phi) =$
$(G_\phi,$ no. of clauses$)$



- Is f poly-time computable?
- YES maps to YES?
  - **Choose 1 true literal per clause in satisfying assignment**
  - **choose corresponding vertices (1 per triangle)**
  - **IS, since no contradictory literals in assignment**

$$\phi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

$f(\phi) =$
$(G,$ no. of clauses$)$



- NO maps to NO? Show if a 3-CNF maps to YES, then satisfiable:
  - **IS can have at most 1 vertex per triangle**
  - **IS of size $\geq$ no of clauses must have exactly 1 per triangle**
  - **since IS, no contradictory vertices**
  - **can produce satisfying assignment by setting these literals to true**

NP-completeness of CLIQUE

Recall that (G, k) is an instance of CLIQUE if G is a graph having k vertices that are all connected to each other. Easy to check that CLIQUE is in **NP**.

NP-completeness of CLIQUE

Recall that (G, k) is an instance of CLIQUE if G is a graph having k vertices that are all connected to each other. Easy to check that CLIQUE is in **NP**.

NP-hard: Reduce from INDEPENDENT SET
Given $G = (V, E)$ and number k (for which, we ask whether (G, k) is an instance of INDEPENDENT SET), construct $G' = (V', E')$ and number k' such that (G, k) has an independent set if and only if (G', k') has a clique of size k'.

NP-completeness of CLIQUE

Recall that (G, k) is an instance of CLIQUE if G is a graph having k vertices that are all connected to each other. Easy to check that CLIQUE is in **NP**.

NP-hard: Reduce from INDEPENDENT SET
Given $G = (V, E)$ and number k (for which, we ask whether (G, k) is an instance of INDEPENDENT SET), construct $G' = (V', E')$ and number k' such that (G, k) has an independent set if and only if (G', k') has a clique of size k'.

Switch edges and non-edges — a size-k independent set becomes a size-k clique. (So, let $k' = k$.) A size-k set that is not independent fails to become a size-k clique!

Another reduction: 3-SAT $\leq_P$ READ-5-TIMES 3-SAT

Definition

READ-5-TIMES 3-SAT consists of 3-CNF formulae where any propositional variable can appear at most 5 times.

Suppose that variable x appears r times in ϕ.
Replace the i-th occurrence with x_i $(1 \leq i \leq r)$ and add new clauses:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge \ldots$$

$$(x_{r-1} \vee \neg x_r) \wedge (\neg x_{r-1} \vee x_r) \wedge$$

The new clauses require $x_1, \ldots, x_r$ to have the same truth value, in any satisfying assignment. It is not hard to check that the new formula can be constructed in polynomial time.

Another reduction: LATIN SQUARE COMPLETION $\leq_P$ SUDOKU

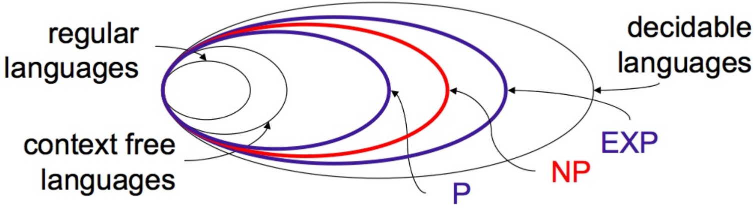
An instance of SUDOKU is a $n \times n$ grid of $n \times n$ sub-grids, some entries containing numbers in the range $1, \ldots, n^2$; it is a YES-instance if it has a solution (i.e., you can fill in all entries so that all numbers in a subgrid are distinct, and all numbers in a row or column are distinct.

An instance of LATIN SQUARE COMPLETION is a $n \times n$ grid, some entries with numbers in range $1, \ldots, n$; it's a YES-instance if it can be filled with numbers in the range $1, \ldots, n$ such that all numbers in any row, and all numbers in any column are distinct.

http://www.dcs.warwick.ac.uk/~czumaj/cs301/PGoldberg/sudoku.html

It's easy to prove SUDOKU **NP**-complete... if you happen to know already that LATIN SQUARE COMPLETION is **NP**-complete!

Complexity Summary



We do not if **P** $\neq$ **NP**, or **NP** $\neq$ **EXP**
We do know know how to prove lots of interesting problems are **NP**-hard "presumably intractable".
In the exercises, do more examples.