TIME COMPLEXITY

Measuring computational complexity

- Time is the most limiting resource
- Computation time = number of steps

  = number of cfgs in computation trace
- Steps on a ***Turing machine***

  which faithfully counts moves, as do physical devices

Asymptotic complexity

- Performance of algorithms may differ wildly for different inputs.
- Measure complexity by bound on resources consumed
  as a function of input *size* ("worst-case complexity").
- For a TM M over Σ let $T_M(w)$ be
  the number of cfg's in the trace of M for input $w \in \Sigma^*$.
  This is defined only if M terminates on w.
- TM M $\boxed{\textit{runs within time } f}$ ($f : \mathbb{N} \to \mathbb{N}$)
  if $T_M(w) \leqslant f(|w|)$ for all inputs w.
- So if M runs within f and $f \leqslant g$
  then M runs within g as well.

Which machine model

- You might take issue with using Turing machines as reference.

  TMs “don’t cheat”, but perhaps they are too simple.

- For example, to compute the function $w \mapsto w \cdot w$ (doubling the input)

  A Turing transducer moves each symbol in w a distance w ,

  so the computation take $> |w|^2$ steps.

- If we use an auxiliary string (“tape”) the doubling of w

  can be performed in $c \cdot |w|$ steps, for some small constant c.

Which machine model

- You might take issue with using Turing machines as reference.

  TMs “don’t cheat”, but perhaps they are too simple.

- For example, to compute the function $w \mapsto w \cdot w$ (doubling the input)

  A Turing transducer moves each symbol in w a distance w,

  so the computation take $> |w|^2$ steps.

- If we use an auxiliary string (“tape”) the doubling of w

  can be performed in $c \cdot |w|$ steps, for some small constant c.

- Useful generalization of Turing machines:

  multi-tape Turing machines,

  each using a fixed number of strings.

Comparing asymptotic behaviors

- Asymptotic behavior of a function $f : \mathbb{N} \to \mathbb{N}$:

  behavior "at infinity", for large arguments growing yet larger.

- Example: $10 \cdot n^3 < 2^n$ for all "sufficiently large" n (here $n > 15$).

Comparing asymptotic behaviors

- Asymptotic behavior of a function $f : \mathbb{N} \rightarrow \mathbb{N}$:

  behavior "at infinity", for large arguments growing yet larger.

- Example: $10 \cdot n^3 < 2^n$ for all "sufficiently large" n (here $n > 15$).

- An *asymptote* (of a curve) in geometry

  is a line tangent to a curve at infinity.

- Example: The x-axis is an asymptote of the curve $y = 1/x$.

  So is the y-axis.

Comparing asymptotic behaviors

- Asymptotic behavior of a function $f : \mathbb{N} \to \mathbb{N}$:

  behavior "at infinity", for large arguments growing yet larger.

- Example: $10 \cdot n^3 < 2^n$ for all "sufficiently large" n (here $n > 15$).

- An *asymptote* (of a curve) in geometry

  is a line tangent to a curve at infinity.

- Example: The x-axis is an asymptote of the curve $y = 1/x$.

  So is the y-axis.

- `a-syn-ptote` Greek for *not falling together*

Coefficients ignored: big-O notation

- Circustantial details may double or triple machine performance.

  It makes sense to abstract away from such details.

- Define $f \preccurlyeq g$ if for some $c > 0$ we have

  $f(n) \leqslant c \cdot g(n)$ for all sufficiently large n.

  I.e. there is some k s.t. $f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

Coefficients ignored: big-O notation

- Circustantial details may double or triple machine performance.

  It makes sense to abstract away from such details.

- Define $f \preccurlyeq g$ if for some $c > 0$ we have

  $f(n) \leqslant c \cdot g(n)$ for all sufficiently large n.

  I.e. there is some k s.t. $f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

- We say then that "*f is* $\boxed{\textit{big-O}}$ *of g*"

  and write $f = O(g)$.

- More accurately we could define $O(g) = \{f \mid f \preccurlyeq g\}$

  and then write $f \in O(g)$.

Coefficients ignored: big-O notation

- Circustantial details may double or triple machine performance.

  It makes sense to abstract away from such details.

- Define $f \preccurlyeq g$ if for some $c > 0$ we have

  $f(n) \leqslant c \cdot g(n)$ for all sufficiently large n.

  I.e. there is some k s.t. $f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

- We say then that *"f is **big-O** of g"*

  and write $f = O(g)$.

- More accurately we could define $O(g) = \{f \mid f \preccurlyeq g\}$

  and then write $f \in O(g)$.

- $f = O(g)$ means $\exists c, k\ n\ \ f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

  or $\exists c \forall^{\infty} n\ \ f(n) \leqslant c \cdot g(n)$.

Coefficients ignored: big-O notation

- Circustantial details may double or triple machine performance.

  It makes sense to abstract away from such details.

- Define $f \preccurlyeq g$ if for some $c > 0$ we have

  $f(n) \leqslant c \cdot g(n)$ for all sufficiently large n.

  I.e. there is some k s.t. $f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

- We say then that "*f is* $\boxed{\textbf{big-O}}$ *of g*"

  and write $f = O(g)$.

- More accurately we could define $O(g) = \{f \mid f \preccurlyeq g\}$

  and then write $f \in O(g)$.

- $f = O(g)$ means $\exists\, c, k\; n \;\; f(n) \leqslant c \cdot g(n)$ for all $n \geqslant k$.

  or $\exists c \forall^{\infty} n \;\; f(n) \leqslant c \cdot g(n)$.

- Convention: use n as a catch-all variable for natural numbers,

  writing eg $O(n^2)$ for "$O(f)$ where $f(n) = n^2$."

⋆*Other asymptotic behaviors*

$\boxed{f = O(g)}$

$\exists c\, \forall^\infty n \quad f(n) \leqslant c \cdot g(n)$

f/g is bounded from above

$\boxed{f = \Omega(g)}$

$\exists c\, \forall^\infty n \quad f(n) \geqslant c \cdot g(n)$

f/g is bounded from below

$\boxed{f = o(g)}$

$\forall c\, \forall^\infty n \quad f(n) \leqslant c \cdot g(n)$

$f/g \to 0$

$\boxed{f = \omega(g)}$

$\forall c\ \forall^\infty n \quad f(n) \geqslant c \cdot g(n)$

$f/g \to \infty$

$\boxed{f = \Theta(g)}$

$\exists\, c, c'\ \forall^\infty\, n \qquad c \cdot g(n) \leqslant f(n) \leqslant c' \cdot g(n)$

g & f have similar asymptotic behavior

Time complexity classes

- TM M $\boxed{\textit{runs in time } O(f) \textit{ (“order } f\textit{”)}}$

  if its time complexity is $c \cdot f$

  for some constant $c > 0$.

- The f's of interest are non-decreasing:

  $f(n+1) \geqslant f(n)$ for all n.

- Examples: $\log n,\ n,\ n \log n,\ n^2,\ n^5,\ 2^n,\ 2^{n^2},\ n!,\ n^n$.

- We write **Time**(f) for the collection of languages

  recognized by a Turing acceptor in time $O(f)$.

- Similar notation for transducers.

- The reference to Turing machines is needed:

  if another machine model is used then we needs to specify,

  as in “this algorithms runs in quadratic time on a RAM.

The Time Hierarchy Theorem

- We can expect that significantly more computation time implies that more functions are computable.
- This is indeed true in virtually all practical cases:
- ***Time Hierarchy Theorem.*** *If*

  (1) $t, T : \mathbb{N} \to \mathbb{N}$ *are "reasonable"; and*

  (2) $t(n)\log(t(n)) = o(T(n))$ *then* **Time** $(t) \subsetneq$ **Time** (T).
- Using Calculus notations, the main condition of the Theorem states

$$\frac{t(n) \cdot \log(t(n))}{T(n)} \to 0 \quad (n \to \infty)$$

Using the Time-Hierarchy Theorem

- A function f is “reasonable” means here that

  computable (for unary numerals) in time $O(f)$:

  $f(1^n)$ is computable in a number of steps linear in $f(n)$. Such functions are called ***time-constructible.***

- The time-constructibility condition is essential:

  without it there are huge “gaps”: a lot more computation time

  without obtaining new functions (the **Gap Theorem**).

- **Time**$(n) \subsetneq$ **Time**$(n^2) \subsetneq$ **Time**$(n^3) \subsetneq$ **Time**$(2^n) \subsetneq$ **Time**$(3^n) \subsetneq$ **Time**(2^{n^2})

- But ***not*** **Time**$(n) \neq$ **Time**$(n \cdot \log n)$

  which requires a separate proof.

⋆*Time Hierarchy proof idea*

- Given total functions $f_1, f_2, \ldots$ over $\mathbb{N}$,

  obtain a function g not listed: $g(n) = f_n(n) + 1$.

- Proof idea for Time Hierarchy:

  Part A: List $\mathbf{Time}(t)$, obtain g not in the list.

  Part B: Build a universal interpreter for $\mathbf{Time}(t)$,

  that runs in $\mathbf{Time}(T)$.

  So $g \in \mathbf{Time}(T) - \mathbf{Time}(t)$.

- (B) is technical.

  (A) is thorny: can we list $\mathbf{Time}(t)$?

Listing $\mathbf{Time}(t)$

- The bad news: For any f of interest,

  there is no effective listing of the transducers in time $O(f)$.

- The good news:

  We only need listed a transducer for each *function* in $\mathbf{Time}(t)$.

- For any transducer M , and constant c ,

  define M_c as M with a built-in “clock”,

  aborting computation for input w after $c \cdot f(|w|)$ steps.

- M_c can be made to run in $\mathbf{Time}(t)$, clock and all,

  using the assumption that f is time-constructible.

POLYNOMIAL TIME

Polynomial vs exponential growth rate

- Polynomial growth-rate: $f(n) = n^k$, k fixed.
- Exponential growth-rate: $f(n) = k^n$, k fixed.
- The choice of base k does not change the general picture:

  $p^n = k^{an}$ where $a = \log_k p = \log p / \log k$
- But polynomial and exponential growth-rates tell very different stories:

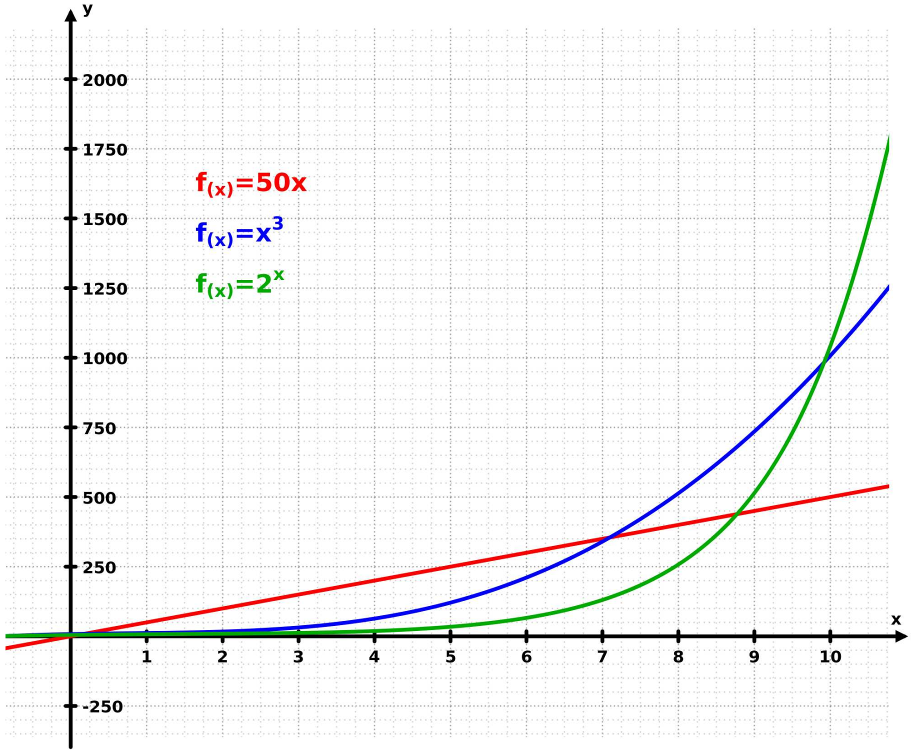
  If an algorithm runs 2^n steps on input of size n, then

  the universe is too small to deal with input of size 300:

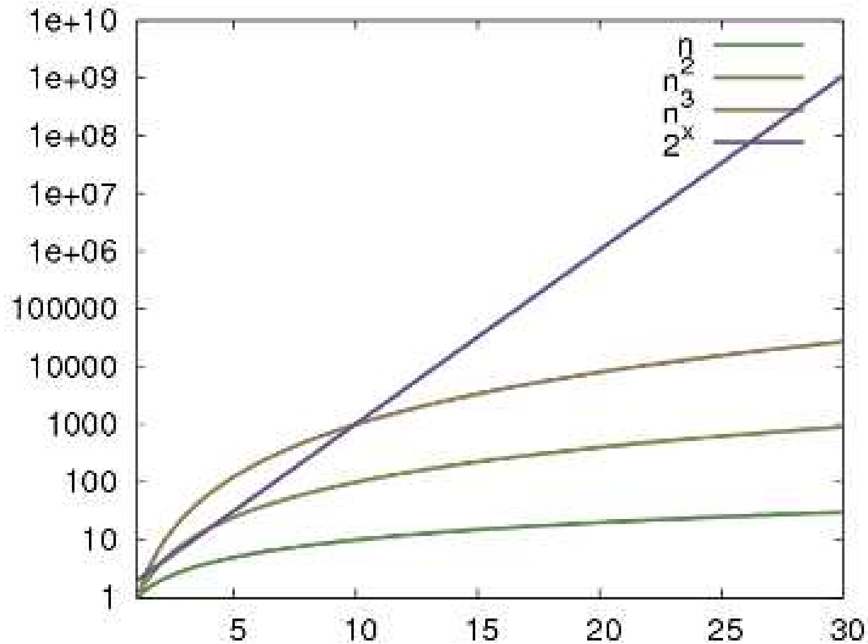
  It is believed that there are $10^{90} \approx 2^{300}$ quarks in the universe.

Graphics

- Any exponential function overtakes any polynomial function for sufficiently large inputs.



• Taking logarithmic scaling for the increase visualizes the difference more clearly:



Every polynomial function flattens out rapidly, whereas any exponential function grows steadily:

$\log(n^k) = k \cdot \log n$, flattening.

$\log(2^n) = n$, steadily increasing

⋆*Exponentials surpass polynomials: an elementary proof*

- **Fact:** For every k we have $2^n > n^k$ for sufficiently large n.
- First, by induction on q we get $2^q > q(q+1)$ for $q \geqslant 5$.
- So if $q \geqslant k \geqslant 5$, then $2^q > q(q+1) \geqslant k(q+1)$.
- Take $n > 2^k$.

  Then $2^q \leqslant n < 2^{q+1}$ for some $q \geqslant k$, and

$$\begin{aligned} 2^n &\geqslant 2^{2^q} && \text{for } n \geqslant 2^q \\ &> 2^{k(q+1)} && \text{since } 2^q > k(q+1) \\ &= (2^{q+1})^k && \\ &> n^k && \text{since } 2^{q+1} > n \end{aligned}$$

**Exponentials surpass polynomials: a calculus proof*

- Write $f \succ g$ for "f eventually exceeds g,"

  i.e. $\exists a\, \forall x > a \quad f(x) > g(x)$.

- By induction on k:

  for every m, $e^x \succ m \cdot x^k$, i.e. $\lim_{x\to\infty} x^k/e^x = 0$

- For $k = 0$ we have $x^0 = 1$, and indeed $\lim_{x\to\infty} 1/e^x = 0$.

- Assuming $\lim_{x\to\infty} x^k/e^x = 0$ we have

$$
\begin{aligned}
\lim_{x\to\infty} x^{k+1}/e^x &= \lim_{x\to\infty} (x^{k+1})'/(e^x)' && \text{by L'Hopital Rule} \\
&= \lim_{x\to\infty} ((k+1)\, x^k)/e^x \\
&= (k+1)\ \lim_{x\to\infty} x^k/e^x \\
&= 0 && \text{by IH}
\end{aligned}
$$

PTime decidable problems

- A Turing decider $\boxed{\textit{runs in polynomial time (PTime)}}$ if its running time on input of size n is $O(n^k)$ for some k.
- All standard machine acceptors can be compiled into Turing machines with increase of computation time bounded by a polynomial (usually n^2).

  So "PTime" remains unchanged from model to model.
- We can therefore consider informal algorithms without worrying about low level implementation.

**The Cobham-Edmunds Thesis*

- PTime is a practical first-approximation
  of the scope of computational ***feasibility***:

Cobham-Edmunds Thesis (1964)

An algorithm is (intuitively) feasible iff it runs in PTime.

- Since all basic computation models simulate each other
  within a factor polynomial in the size of the input,
  this Thesis can refer to “algorithms.”)

Flaws of the Cobham-Edmunds Thesis

- The Cobham-Edmunds Thesis is a declaration of faith, not a theorem, just like the Turing-Church Thesis.

Flaws of the Cobham-Edmunds Thesis

- The Cobham-Edmunds Thesis is a declaration of faith, not a theorem, just like the Turing-Church Thesis.
- But it is far more problematic than the Turing Thesis, and should be taken with a grain of salt, as a rough guide.

Flaws of the Cobham-Edmunds Thesis

- The Cobham-Edmunds Thesis is a declaration of faith, not a theorem, just like the Turing-Church Thesis.
- But it is far more problematic than the Turing Thesis, and should be taken with a grain of salt, as a rough guide.
- Here are some issues that weaken it.

1. The exponents should matter: n^{100} is not feasible.
2. The coefficients should matter: $100^{100} n$ is not feasible.

Flaws of the Cobham-Edmunds Thesis

- The Cobham-Edmunds Thesis is a declaration of faith, not a theorem, just like the Turing-Church Thesis.
- But it is far more problematic than the Turing Thesis, and should be taken with a grain of salt, as a rough guide.
- Here are some issues that weaken it.

1. The exponents should matter: n^{100} is not feasible.
2. The coefficients should matter: $100^{100}\,n$ is not feasible.
3. Conversely, time of order $n^{\log\log n}$ is not admitted, and yet $n^{\log\log n} < n^8$ for all $n < 2^{2^8} = 2^{256} \approx 10^{77}$.

Some important PTime-decidable problems

- CONNECTIVITY: *Given a graph $G = (V, E)$, is it connected?*
- A simple algorithm:

  For each pair u, v of vertices check all permutations

  of the remaining vertices for being a path from u to v.
- This is not feasible, e.g. $100! \approx 10^{158}$.
- But there are algorithms quadratic in the number of nodes. (Dijkstra's Algorithm, 1969)

Other PTime-decidable problem

- LINEAR-INEQUAL: Given a set of linear inequalities,

  does it have a real-number solution?

  Example $3x + y \geqslant 0, \quad x + 3y \leqslant 0$. A PTime decision algorithm was found in 1979 by Leo Khachian.

- EDGE-COVER: Given a graph $\mathcal{G}$ and a target $t > 0$

  is there a set of $\leqslant t$ edges which includes all vertices (Edmunds 1965).

  (In contrast, we know of no PTime-decision for VERTEX-COVER.)

- PRIMALITY: Given a natural number, is it prime?

  A PTime decision algorithm for primality

  was developed in 2006 by Agrawal, Kayal and Saxena.

Enhanced uses of induction

- To reason inductively,

  we sometimes need at each step more than what we wish to prove.

- Example we studied:

  To parse a single (prefix notation) boolean expression,

  parse arbitrary strings into a concatenation of expressions.

Memoization: caching data for repeated use

- $\boxed{\textit{Memoization}}$ = memorize information for future use (Greek: *mnémé* = memory).
- **Example.** If $L \subseteq \Sigma^*$ is PTime decidable then so is L^+.
- How about exhaustive search:
  For each partition of input w into concatenated non-empty substrings check whether all parts are in L.
- There are 2^{n-1} partitions of w of size n!!
- But the number of "parts" is only quadratic in n !
- And (as for the parsing algorithm) we can uniformize matters by finding whether substrings are in L^+ rather than L .
- This we can do by a simple induction on length.

A Ptime algorithm for L^+

- We calculate the set S of substrings of w that are in L^+.
- We do this by induction on length, i.e. calculating successively

  $S_i =$ the set of strings in S of length i.
- S_1 consists of the letters in w.
- S_{i+1} can be calculated from $S_1, \ldots, S_i$ in PTime.
- So the entire algorithm is in PTime.

A concrete case of the algorithm:

- $L =$ English words.
- *letustakethisshortsentenceasaniceexample*
- Consider substrings of length 1. $S_1 =$ *a*
- Substrings of length 2: $S_2 =$ *us,hi,or,as,an,am*
- Substrings of length 3: $S_3 =$ *let,ten,asa,his,ice*
- Substrings of length 4: $S_4 =$ *take,this,hiss,sent*
- Substrings of length 5: $S_5 =$ *letus,stake,short,ample*
- ....
- Substrings of length 37: $S_{37} =$ *ustakethisshortsentenceasaniceexample*
- Substrings of length 39: $S_{39} = \emptyset$
- Substrings of length 40: $S_{40} =$ *letustakethisshortsentenceasaniceexample*

Same idea: CFLs are in **Time**(n^3)

- A useful tool: Chomsky grammars.
- A $\boxed{\textit{Chomsky grammar}}$ is a CFG using only two type of productions:

  [Terminal.] $A \rightarrow \sigma$ $\quad (\sigma \in \Sigma)$

  [Split.] $A \rightarrow BC$ $\quad (B, C$ other than $S)$

- **Theorem**

  Every CFL without ε is generated by a Chomsky grammar.

Cubic-time decidability of CFLs

- Given a Chomsky grammar G over Σ

  we construct a cubic-time memoization algorithm

  deciding whether a given string w is generated by G.

- This is known as the ***Cocke-Younger-Kasami (CYK)*** Algorithm,

  after three who re-discovered it in 1965/67.

  But it was first invented by Itiroo Sakai in 1961!

The CYK Algorithm

- For each non-terminal A of G let

  S_i be the set of pairs SMS(A,u) where

  1. A is a non-terminal,
  2. u a substring of w of length i, and
  3. $A \Rightarrow_G^* u$.

- S_1 is obtained directly from the Unit Productions of G.

Inductive calculation of S_i

- S_{i+1} is obtained from S_j for $j \leqslant i$:

  1. For each substring u of length $i+1$
  2. for each split $u = x \cdot y$ (Note: $|x|, |y| \leqslant i$)
  3. for each Split production $A \to BC$

  If $(B, x) \in S_{|x|}$ and $(C, y) \in S_{|y|}$,
  then place (A, u) in S_{i+1}.

- There are $O(n)$ substrings of length $i \leqslant n$,
  and $O(n)$ splits for each substring,
  so for each $i \leqslant n$ the process is in time $O(n^2)$.

- If $|w| = n$ then there are n passes,
  so the entire algorithm is in cubic time.

A CYK Example

Generating $\mathtt{a}^p\mathtt{c}\mathtt{b}^{p+q}\mathtt{c}\mathtt{a}^q$:

$$
\begin{aligned}
S &\rightarrow LR \\
L &\rightarrow \mathtt{a}L\mathtt{b} \mid \mathtt{c} \\
R &\rightarrow \mathtt{b}R\mathtt{a} \mid \mathtt{c}
\end{aligned}
$$

An equivalent Chomsky grammar:

(1) $S \rightarrow LR$ **(2)** $L \rightarrow AM \mid \mathtt{c}$ **(3)** $M \rightarrow LB$ **(4)** $R \rightarrow BN \mid \mathtt{c}$
(5) $N \rightarrow RA$ **(6)** $A \rightarrow \mathtt{a}$ **(7)** $B \rightarrow \mathtt{b}$

Decide whether **`acbbca`** is generated.

Calculating S_i

The grammar:

$$(1)\quad S \to LR \qquad (2)\quad L \to AM \mid \mathtt{c} \qquad (3)\quad M \to LB \qquad (4)\quad R \to BN \mid \mathtt{c}$$

$$(5)\quad N \to RA \qquad (6)\quad A \to \mathtt{a} \qquad (7)\quad B \to \mathtt{b}$$

The sets:

$$S_1: \ A \Rightarrow \mathtt{a}, \quad B \Rightarrow \mathtt{b}, \quad L \Rightarrow \mathtt{c}, \quad R \Rightarrow \mathtt{c}$$

$$S_2: \ M \to LB \Rightarrow^* \mathtt{cb}$$

$$N \to RA \Rightarrow^* \mathtt{ca}$$

$$S_3: \ L \to AM \Rightarrow^* \mathtt{acb}$$

$$R \to BN \Rightarrow^* \mathtt{bca}$$

$$S_4: \ M \to LB \Rightarrow^* \mathtt{acbb}$$

$$S_5: \ \emptyset$$

$$S_6: \ S \to LR \Rightarrow^* \mathtt{acbbca}$$

Closure properties of PTime problems

- Closure under set operations:

  complement, union, intersection.

- Closure under language operations:

  concatenation, plus, star.

Closure properties of PTime functions

- **PTime** is closed under composition:

  Suppose $f, g : \Sigma^* \to \Sigma^*$.

  If $f \in \textbf{Time}(n^k)$ and $g \in \textbf{Time}(n^\ell)$

  then $f \circ g \in \textbf{Time}((n^k)^\ell) = \textbf{Time}(n^{k \cdot \ell})$.

- Suppose transducer T computes f in time $c \cdot n^k$,

  and T' computes g in time $d \cdot n^\ell$.

- Given input $w \in \Sigma^*$,

  T terminates in $\leqslant c|w|^k$ steps,

  and so has an output y of size $\leqslant c \cdot |w|^k$.

- Given y as input,

  T' operates in time $\leqslant d \cdot |y|^\ell$,

  i.e. $\leqslant e \cdot |w|^{k \cdot \ell}$ ($e = d \cdot c^k$)

PTIME REDUCTIONS

Reminder: reductions between problems

- Let $\mathcal{P}$ and $\mathcal{Q}$ be problems.

  A $\boxed{\textit{reduction}}$ of $\mathcal{P}$ to $\mathcal{Q}$ is a function

$$\rho : \textit{Instances}(\mathcal{P}) \to \textit{Instances}(\mathcal{Q})$$

  such that for every instance w of $\mathcal{P}$,

$$w \in \mathcal{P} \quad \textbf{IFF} \quad \rho(w) \in \mathcal{Q}$$

- I.e., to find out whether $w \in \mathcal{P}$ we can find $\rho(w)$,

  and find whether it is in $\mathcal{Q}$.

Reminder: computable reductions

- If ρ is a computable reduction of $\mathcal{P}$ to $\mathcal{Q}$ then we write $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$ and say that $\mathcal{P}$ ***computably-reduces*** to $\mathcal{Q}$.

PTime reductions

- Computable reductions relate the algorithmic solvability of problems.
- PTime reductions relate the ***feasibility*** of problems: a $\boxed{\textit{PTime reduction}}$ of problem $\mathcal{P}$ to problem $\mathcal{Q}$ is a PTime function ρ that maps instances of $\mathcal{P}$ to instances of $\mathcal{Q}$, such that $w \in \mathcal{P}$ iff $\rho(w) \in \mathcal{Q}$.
- We write $\rho : \mathcal{P} \leqslant_p \mathcal{Q}$, with a subscript p.
- If there is such a ρ, we write $\mathcal{P} \leqslant_p \mathcal{Q}$ and say that $\mathcal{P}$ $\boxed{\textit{PTime-reduces}}$ to $\mathcal{Q}$.

Transitivity of PTime-reductions

- We had:

  If $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$ and $\rho' : \mathcal{Q} \leqslant_c \mathcal{R}$

  then $\rho \circ \rho' : \mathcal{P} \leqslant_c \mathcal{R}$

- Since PTime is closed under composition,

  we similarly have:

  If $\rho : \mathcal{P} \leqslant_p \mathcal{Q}$ and $\rho' : \mathcal{Q} \leqslant_p \mathcal{R}$

  then $\rho \circ \rho' : \mathcal{P} \leqslant_p \mathcal{R}$

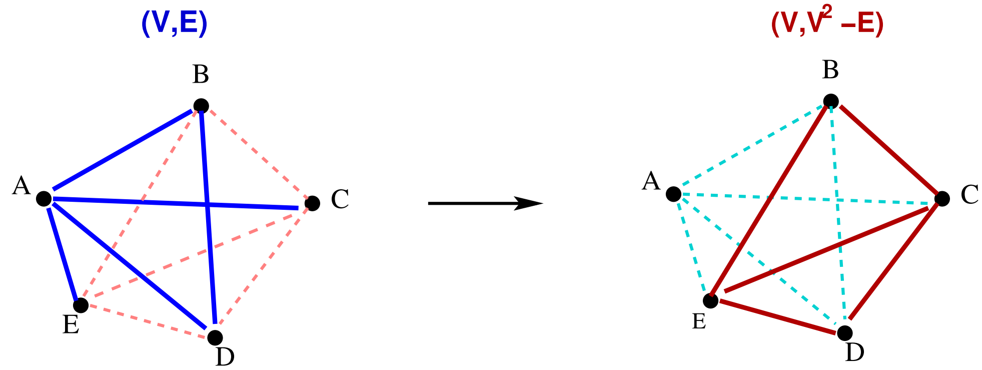
- Note the benefit of lumping together all polynomials:

  For example, reducibility by quadratic time reduction

  is ***not*** closed under composition.

1/2-CLIQUE *reduces to* CLIQUE

Example: CLIQUE *reduces to* INDEPENDENT-SET



A blue graph
Missing edges are in pink
{A,B,D} a clique of size 3

A red graph
Missing edges are in blue
{A,B,D} an ind set of size 3

1/2-CLIQUE *reduces to* 1/3-CLIQUE

CLIQUE *reduces to* 1/2-CLIQUE

HAMILTONIAN-PATH *reduces to* HAMILTONIAN-CYCLE

*EXACT-SUM *reduces to* INTEGER-PARTITION

- Reductions may be ingenious,

  using particulars of the problems compared.

  There are no silver bullets.

- Reducing INTEGER-PARTITION (IP) to EXACT-SUM (ES) was easy,

  because IP is a secial case of ES.

- But we also have $\rho :$ **IP** $\leqslant_p$ **ES**

  by the following PTime reduction ρ:

- Given instance (S,t) of ES let $A = \Sigma S$.

  Note: $t < A$: o/w (S,t) is trivially not in ES.

  Define $S' = \rho(S,t) =_{\rm df} S \cup \{A+t,\ 2A-t\}$. Note: $\Sigma S' = 4A$.

- We show that S has a subset P adding up to t

  iff S' has a subset P' adding up to $(\Sigma S')/2 = 2A$.

- If we have $P \subset S$ with $\Sigma P = t$

  then take $P' = P \cup \{2A - t\}$.

- Conversely, suppose exists some $P' \subset S'$ satisfying $\Sigma P' = 2A$.

  Let P be the one of P' and $S' - P'$ that has $2A - t$.

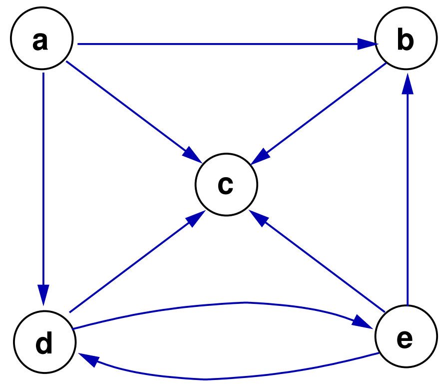
  Since $(A + t) + (2A - t) = 3A$ and $\Sigma P = 2A$

  P cannot have $A + t$.

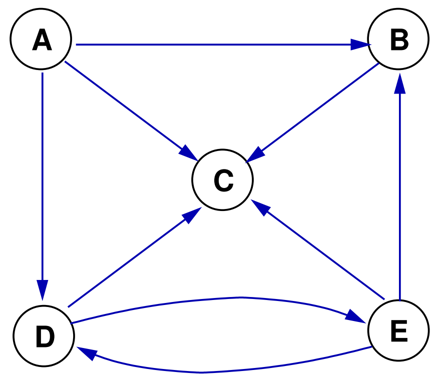
  Let $P = P' - \{2A - t\}$. Then $P \subseteq S$ and $\Sigma P = t$.

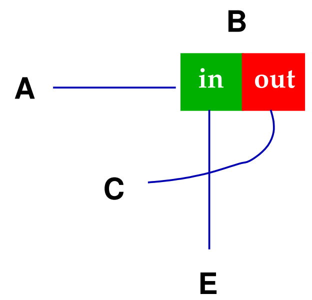
HAMILTONIAN-PATH *reduces to* UNDIRECTED-HAMILTONIAN-PATH

- Does this directed-graph have a Hamiltonian path?

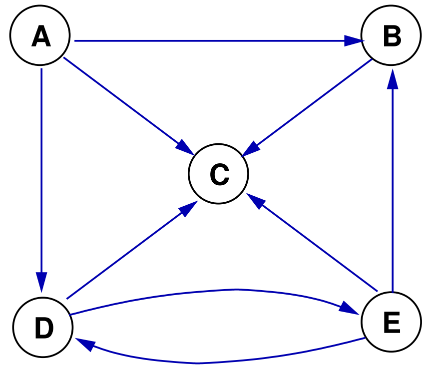


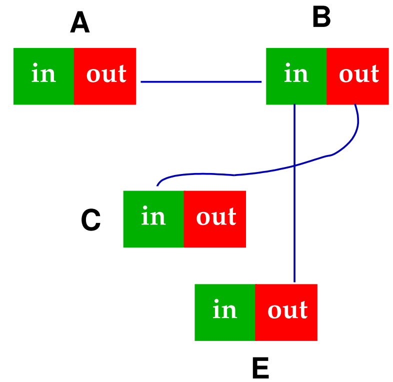
• Creating a direction-gadget:





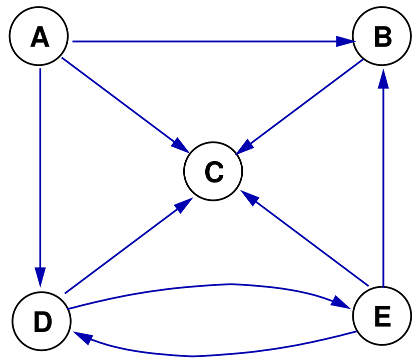
- Same for the neighboring

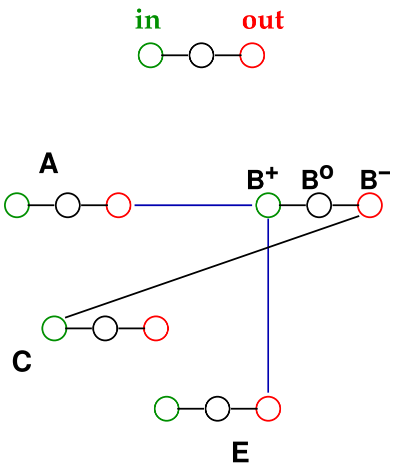




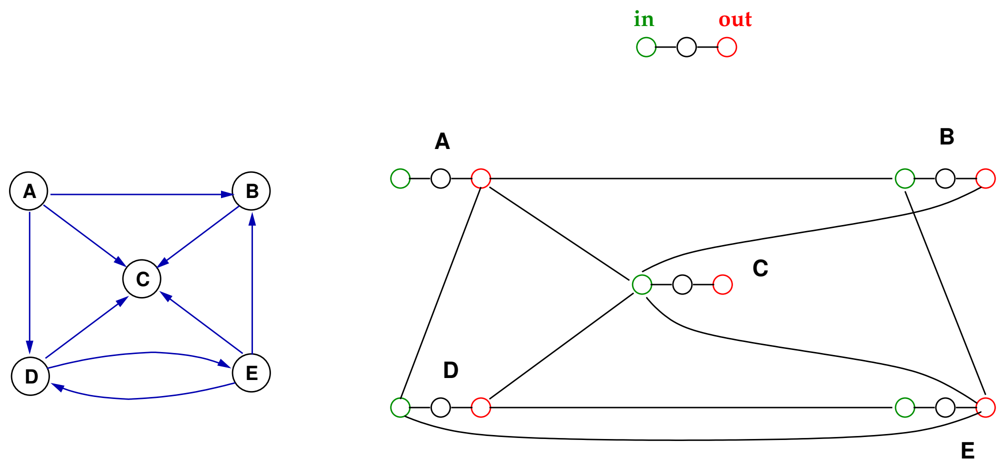
- But is directionality assured?

- We need a middle-node in each gadget:





- For the entire graph:



- This mapping is a PTime reduction of HAMILTONIAN-PATH to UNDIRECTED-HAMILTONIAN-PATH.

PTime reductions and problem feasibility

- Had: If $\mathcal{Q}$ is decidable and $\mathcal{P} \leqslant_c \mathcal{Q}$

  then $\mathcal{P}$ is decidable.

- Now: If $\mathcal{Q}$ is ***PTime***-decidable and $\mathcal{P} \leqslant_p \mathcal{Q}$

  then $\mathcal{P}$ is ***PTime***-decidable.

- I.e., If $\mathcal{P}$ is ***not*** PTime-decidable and $\mathcal{P} \leqslant_p \mathcal{Q}$

  then $\mathcal{Q}$ is ***not*** PTime-decidable.

- Similarly: If $\mathcal{Q}$ is **NP** and $\mathcal{P} \leqslant_p \mathcal{Q}$ then $\mathcal{P}$ is **NP**

- I.e.: If $\mathcal{P}$ is ***not* NP** and $\mathcal{P} \leqslant_p \mathcal{Q}$ then $\mathcal{Q}$ is ***not* NP**

Examples

- INTEGER-PARTITION $\leqslant_p$ EXACT-SUM.

  So if EXACT-SUM is PTime-decidable then so is INTEGER-PARTITION.

- CLIQUE $\leqslant_p$ INDEPENDENT-SET.

  So if INDEPENDENT-SET is PTime-decidable then so is CLIQUE.

PTime reduces to linear time

- The Time Hierarchy Theorem implies that for every $k > 0$ there are problems decidable in time $O(n^{k+1})$ but not in time $O(n^k)$.
- But the distinction between the powers in PTime is
  obliterated by PTime reductions.
- Suppose problem $\mathcal{P}$ is decidable by M within time $a \cdot n^k$, for $n > h$.
  Then it has a variation $\mathcal{P}'$ s.t. $\mathcal{P} \leqslant_p \mathcal{P}'$
  but $\mathcal{P}'$ is decidable for ***all*** w in time $\leqslant |w|$.
- Let $\mathcal{P}' = \{ \ w \cdot \sqcup^{m_w} \ \mid \ w \in \mathcal{P}, m_w = a \cdot |w|^k + H \ \}$
  where $H = \max\{\textsf{Time}_M(x) \ \mid \ |x| \leqslant h\}$.
- Define $\rho : \ \mathcal{P} \leqslant \mathcal{P}'$ by $\rho(w) = w \cdot \sqcup^{m_w}$.
- ρ is computable, and is a reduction by defn of $\mathcal{P}'$.
- But $\mathcal{P}'$ is decidable in time identical to the length of the input.

Another simplification

- Suppose a problem $\mathcal{P}$ is deidable within time $a \cdot n$
  for all $n \geqslant k$.
  There is a problem $\mathcal{P}'$,
  decidable within time n for ***all*** input,
  such that $\mathcal{P} \leqslant_p \mathcal{P}'$.
- The proof is similar to the one above: Use padding.

PTIME CERTIFICATION

Exhaustive search for real-life problems

- Some problems require exponential time algorithms because exponentiation is explicit in their specification.
- Transducer example (exponentially large output):
  For input w output a string of length $\geqslant |w|$.

  Acceptor example (exponentially long trace):
  Given acceptor M and string w,
  does M runs $\geqslant 2^{|w|}$ steps on input w ?
- However our examples of exhaustive search are unrealistic because the ***number of cases*** is forbidding,
  not because the specification is unrealistic!

Reminder: Certifications

- A ***certification*** for a decision problem $\mathcal{P}$

  is a binary relation $\vdash_P$ between strings (the ***certificates***),

  and instances of $\mathcal{P}$, such that for all instances w

$$w \text{ satisfies } \mathcal{P} \quad \text{IFF} \quad c \vdash_P w \quad \text{for some} \quad c \in C$$

- We showed that a language L is SD

  iff it has a decidable certification.

Feasible-certification

- A certification $\vdash$ for $\mathcal{P}$ is $\boxed{\textit{feasible}}$
  if $c \vdash w$ is decidable in time polynomial in $|w|$.
  We write then $c \vdash_p w$.
- In time t a Turing acceptor cannot read more
  than the t initial symbols of c, so $c \vdash w$ implies that
  $|c|$ is eventually bounded
  by $|w|^k$ for some k.
- Conversely, if the truth of $c \vdash w$ is computable in time
  polynomial in $|w| + |c|$,
  and $|c|$ is bounded by a polynomial in $|w|$,
  then $c \vdash w$ is PTime in $|w| + |c|$, i.e. PTime as a set.

- In summary: $\mathcal{P}$ is feasibly certified iff $c \vdash w$ is decidable in PTime (w, c both counted!)

  and $|c|$ is bounded by a polynomial in $|w|$.

- Restricting certificate size is essential:

  otherwise any SD problem $\mathcal{P}$ would be PTime certified

  because the time to check that a trace c is correct is $O(|c|^2)$,

  and so is polynomial in $|w| + |c|$.

NP: Non-deterministic PTime

- The class of PTime-certified problems is also referred to as $\boxed{\textit{NP}}$ short for *"Non-deterministic PTime"*.
  The reasons are mostly of historical interest.
- A non-deterministic (ND) Turing acceptor is defined like an acceptor, expect that its transition mapping is not necessarily univalent.
- We say that an ND acceptor M $\boxed{\textit{accepts a string } w}$ if there is an accepting computation-trace c of M for input w.
- Moreover, that acceptance is $\boxed{\textit{within time } \leqslant t}$ if the trace c has $\leqslant t$ cfgs.
- M is PTime if there are $a, k, h > 0$ such that if M accepts w, $|w| > h$, then it accepts w in time $\leqslant a \cdot |w|^k)$.
- A language L is in **NP** if it is recognized by a ND PTime acceptor.

NP = Feasibly certified

- Feasible certification for a language L implies a non-deterministic recognizing algorithm:
  - A problem $\mathcal{P}$ with a feasible certification is recognized in PTime by a "non-deterministic algorithm":
  - Given an instance w, guess a certificate c.
    This takes time $|c|$, i.e. polynomial in $|w|$.
  - Checking $c \vdash w$ takes time polynomial in $|w|$, since $\vdash$ is feasible.
- Conversely, recognition of L by a PTime ND algorithm implies that L is feasibly certified:
  - Suppose $\mathcal{P}$ is recognized in PTime by an ND algorithm M.
  - A certificate for an instance w is any road map that steers the ND choices to an accepting trace.

NP-COMPLETENESS:
Maximally complex NP problems

Maximal complexity in SD

- A problem $\mathcal{P}$ is $\boxed{\textit{SD-hard}}$

  if every SD problem is computably-reducible to $\mathcal{P}$.

- If $\mathcal{P}$ is SD-hard, and $\mathcal{P} \leqslant_c \mathcal{P}'$ then $\mathcal{P}'$ is SD-hard:

  Every SD problem $\mathcal{Q}$ is reducible to $\mathcal{P}$ since $\mathcal{P}$ is SD-hard.

  So by transitivity of $\leqslant_c$ it follows that $\mathcal{P} \leqslant_c \mathcal{P}'$ we get by $\mathcal{Q} \leqslant_c \mathcal{P}'$.

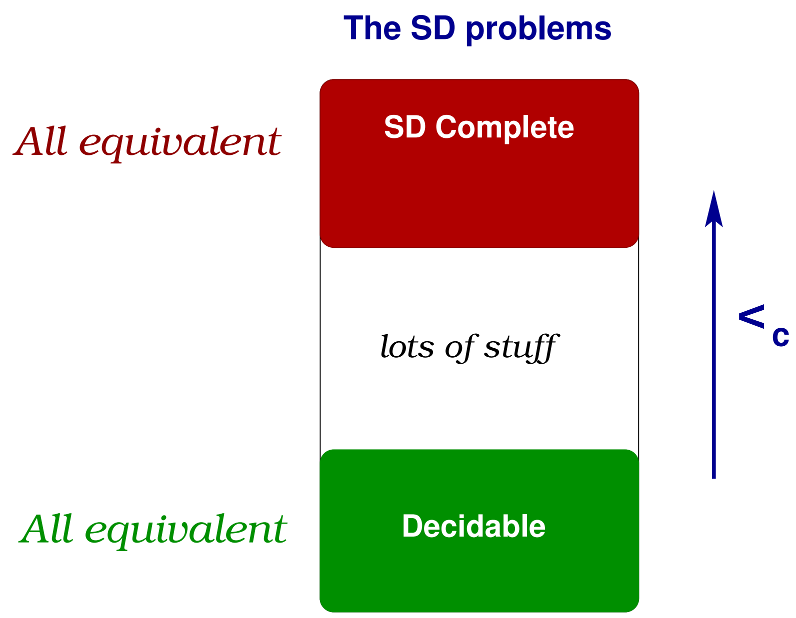
- $\mathcal{P}$ is $\boxed{\textit{SD-complete}}$ if it is SD-hard and is itself SD.

- An obvious SD-complete problem: ACCETANCE.

  If $\mathcal{P} = \mathcal{L}(M)$ then $\mathcal{P} \leqslant_c$ **ACCEPTANCE** by a reduction that maps instance w of $\mathcal{P}$

  to the instance $(M^{\#}, w)$ of accept.

Clear broad picture for SD...



Maximally complex NP problems

- A problem $\mathcal{P}$ is $\boxed{\textit{NP-hard}}$ if every problem in **NP** is $\leqslant_p$ $\mathcal{P}$.
- Since $\leqslant_p$ is transitive, if $\mathcal{P}$ is NP-hard, and $\mathcal{P} \leqslant_p \mathcal{P}'$,

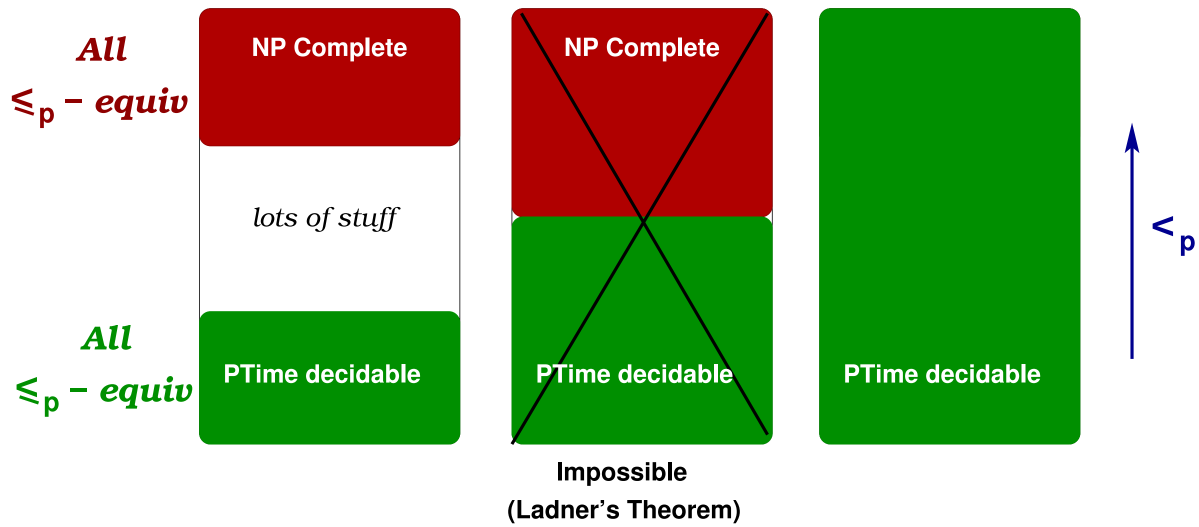
  then $\mathcal{P}'$ is NP-hard as well.
- A problem $\mathcal{P}$ is $\boxed{\textit{NP-complete}}$ if it is both NP and NP-hard.
- From these definitions it follows that if there is

  an NP-hard problem $\mathcal{P}$ which is PTime decidable,

  then every NP problem is PTime-decidable!

Blurry picture for NP

The NP problems: 2 possibilities



Computing is binary...

- We conceive a certification $\vdash_{\mathcal{P}}$ for a problem $\mathcal{P}$ in two stages:

  1. Identify what sort of objects are the certificates.

     E.g. a certificate for an instance of HAMILTONIAN-PATH

     is a list ℓ without repetition of the vertices.

  2. State properties that make a certificate valid.

     For HAMILTONIAN-PATH these are:

     ℓ is without repetitions, and

     successive entries are adjacent in G .

Reminder: Boolean valuations

- Boolean expressions are generated from variables
  using negation, conjunction, and disjunction.
  Example: $(-x) \wedge -(y \vee x)$.
- Given a valuation $V : Var \to \{0,1\}$ of variables,
  each boolean expression evaluates to 0 or 1.
- Example: If $V(x) = 0,\ V(y) = 0$ then $V(-x \wedge -(y \vee x)) = 1$
- A valuation V ***verifies*** E if $V(E) = 1$.
- E is ***satisfiable*** if it is verified by ***some*** V,
  It is ***valid*** if it verified by ***every*** V.
- So E is satisfiable iff $-E$ is not satisfiable
  and is valid iff $-E$ is not satisfiable.

Boolean satisfiability

- BOOL-SAT: Is a given boolean expression satisfiable?
- A certification for BOOL-SAT:

  A certificate for an expression E is a valuation verifying it.
- Checking a certificate is PTime in the size of the expresson.

  So the certification is feasible.

Coding certificates by boolean expressions

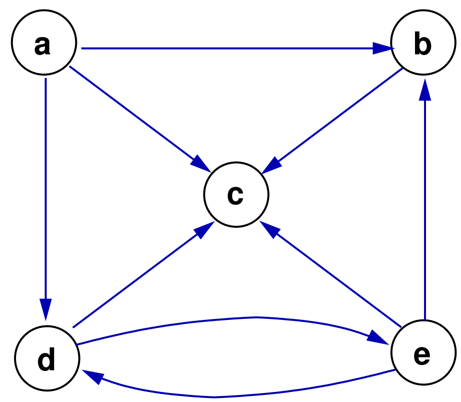
- Digital coding is central to describing discrete data,
  and the simplest form of digital coding is binary, i.e. using booleans.
- No surprise then that a good candidate for NP-hardness
  is Boolean Satisfiability bool-sat.
- We use yes/no questions to code the potential certificates,
  and then yes/no questions that check their validity as certificates.

Boolean coding of potential certificates

- Let's look again at the HAMILTONIAN-PATH (HP) Problem:
  Does a given directed graph $G = (V, E)$ have a Hamiltonian path?
- Let n be the number of vertices in G.
  The question is: Is there a listing
  $u_1, u_2, \ldots, u_n$ of all vertices, without repetition,
  so that $u_i(E)u_{i+1}$ for $i < n$.
- We convey this intent by a boolean expression, using
  for each $v \in V$ and $i = 1..n$ a fresh boolean variable x_{iv}
  intended to be true iff the i'th entry in the list is v.

Using booleans to state the existence of a H-path

- Given G, we construct a boolean expression E_G stating that the boolean variables x_{iv} describe a Hamiltonian path.
- This will show that G has a Hamiltonian path iff E_G is satisfiable.
- For cocreteness, consider our earlier example:



- Any listing in positions 1,2,3,4,5 of the vertices $V = \{a, b, c, d, e\}$ will assign truth values for the 25 variables.

- E.g. the listing a, b, c, d, e is conveyed by the valuation assigning 1 to $x_{1a},\ x_{2b},\ x_{3c},\ x_{4d},\ x_{5e}$ and 0 to the remaining 20 variables.

  Here is that valuation, with the variable set to 1 (true) in orange.

$$
\begin{array}{ccccc}
x_{1a} & x_{1b} & x_{1c} & x_{1d} & x_{1e} \\
x_{2a} & x_{2b} & x_{2c} & x_{2d} & x_{2e} \\
x_{3a} & x_{3b} & x_{3c} & x_{3d} & x_{3e} \\
x_{4a} & x_{4b} & x_{4c} & x_{4d} & x_{4e} \\
x_{5a} & x_{5b} & x_{5c} & x_{5d} & x_{5e}
\end{array}
$$

- Our Hamiltonian path, $a \to d \to e \to b \to c$: is conveyed by the following valuation:

$$
\begin{array}{ccccc}
x_{1a} & x_{1b} & x_{1c} & x_{1d} & x_{1e} \\
x_{2a} & x_{2b} & x_{2c} & x_{2d} & x_{2e} \\
x_{3a} & x_{3b} & x_{3c} & x_{3d} & x_{3e} \\
x_{4a} & x_{4b} & x_{4c} & x_{4d} & x_{4e} \\
x_{5a} & x_{5b} & x_{5c} & x_{5d} & x_{5e}
\end{array}
$$

The vertex-listing is a path

- We state the conditions that make a valuation of the variables x_{iv} into a Hamiltonian path.
- At least one position per vertex:
  For each vertex v the disjunction $x_{1v} \vee \cdots \vee x_{nv}$.
- At most one position per vertex:
  For each vertex v and distinct $i, j = 1..n$
  the expression $-(x_{iv} \wedge x_{jv})$

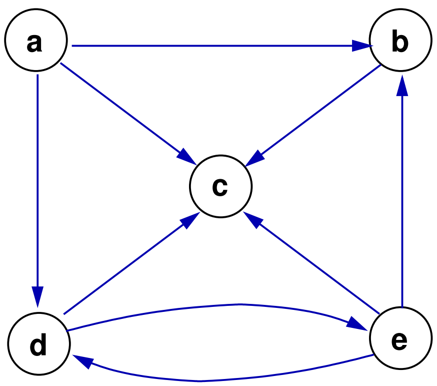
Successive vertices are adjacent in the graph

- For each position $i < n$

  the disjunction of all expressions $x_{iv} \wedge x_{i+1,u}$ where $v(E)u$.

- E.g., positions 2 and 3 are related by one of the 9 edges:

$$
\begin{aligned}
&(x_{2a} \wedge x_{3b}) \vee (x_{2a} \wedge x_{3c}) \vee (x_{2a} \wedge x_{3d}) \\
&\quad \vee (x_{2b} \wedge x_{3c}) \\
&\quad \vee (x_{2d} \wedge x_{3c}) \vee (x_{2d} \wedge x_{3e}) \\
&\quad \vee (x_{2e} \wedge x_{3b}) \vee (x_{2e} \wedge x_{3c}) \\
&\quad \vee (x_{2e} \wedge x_{3d})
\end{aligned}
$$



The reduction

- We've obtained a reduction $\rho:$ **HP** $\leqslant_p$ **BOOL-SAT**
- ρ maps a directed grarph $G = (V, E)$ to the conjunction A_G of the boolean expressions as above,

  based on the particular size and edge-relation of G.
- A_G is computable in time cubic in the size of G.

- The mapping ρ is a reduction:
  - If there is a Hamilt path $u_1 \to \cdots \to u_n$ in G
    then the boolean expression A_G is satisfied by the valuation
    that assigns 1 to x_{iv} iff v is u_i.
  - Conversely, if the expression A_G is satisfied by a valuation V
    then $(v_1..v_k)$ is a Hamilt path,
    where v_i is the unique v for which $V(x_{iv}) = 1$.
- Conclusion: $\rho :$ **HAMILT-PATH** $\leqslant_p$ **BOOL-SAT**

From ND PTime to ND linear time

- We show that every problem
  recognized by a ND acceptor M in PTime $\leqslant_p$ BOOL-SAT.
- The method is similar to the boolean coding of HAMILONIAN-PATH.
- We saw that each problem decidable in PTime
  is PTime-reducible to a problem decidable on site.
- The same padding technique shows that
  each problem recognized by a ND acceptor in PTime
  is PTime reducible to a problem recognized by a ND acceptor on-site.
- By transitivity of $\leqslant_p$ we only need ONSITE-ACCEPT $\leqslant_p$ BOOL-SAT.
-

Coding ND on-site acceptor in BOOL-SAT

- Define a PTime reduction $\rho :$ ONSITE-ACCEPT $\leqslant_p$ BOOL-SAT.
- ρ maps (M, w) (M a ND) to bool expssn $E_{M,w}$ s.t.

  M accepts w in time $|w|$ iff $E_{M,w}$ is satisfiable.

- The trace in grid form:

state									
a	>	1	1	0	1	_	_	_	_
q	>	1	1	0	1	_	_	_	_
q	>	1	1	0	1	_	_	_	_
q	>	1	1	0	1	_	_	_	_
p	>	1	1	0	1	_	_	_	_
r	>	1	1	0	1	_	_	_	_
r	>	0	1	0	1	_	_	_	_
s	>	0	1	0	1	_	_	_	_

|w|

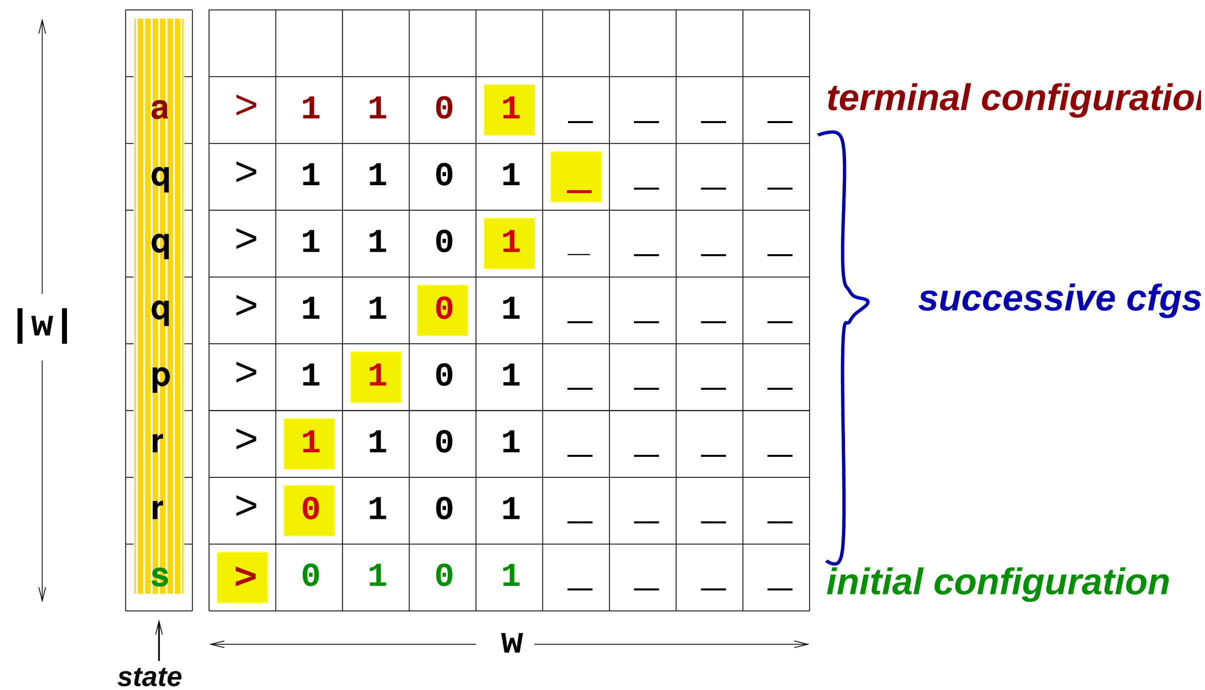
w

terminal configuratio

successive cfgs

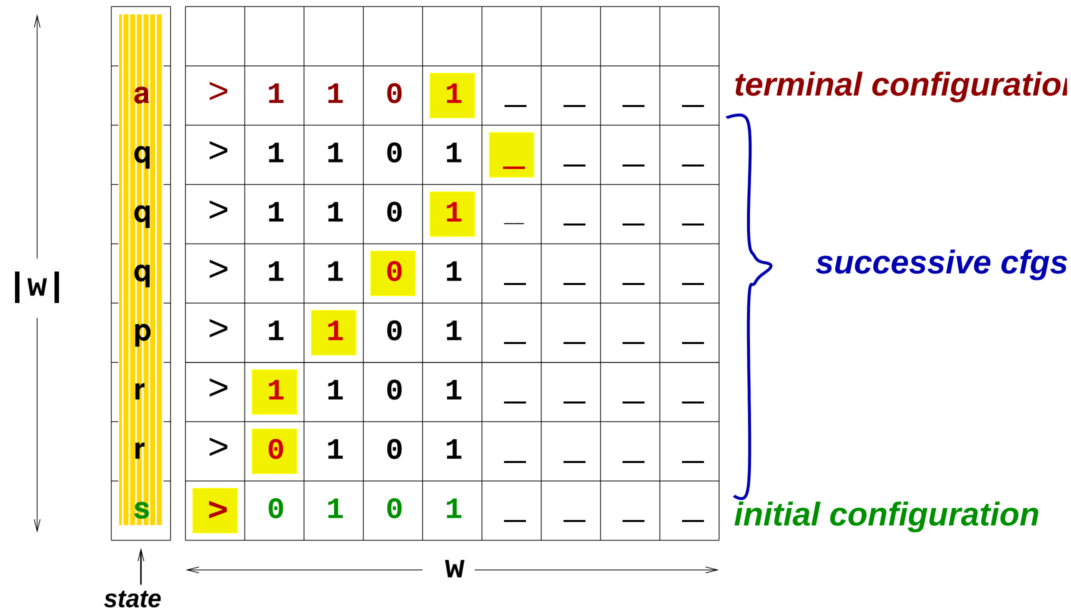
initial configuration

The grid as yes/no questions



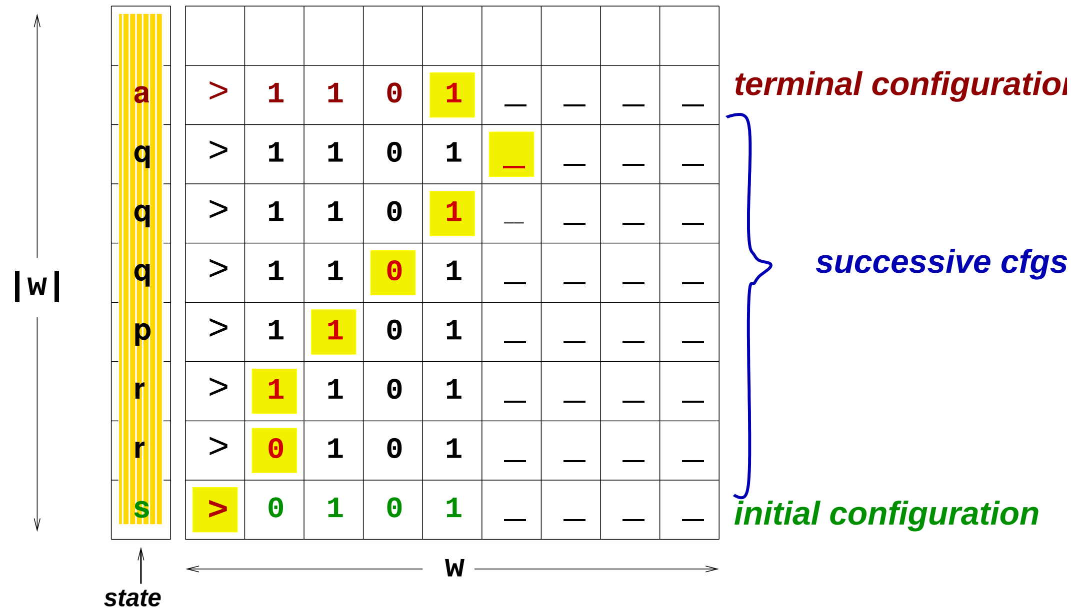
- For each state q and $i \leqslant |w|$ $\;x_{i,q}$ for *“state of i’th cfg is q”*
- For each $i, j \leqslant |w|$: $c_{i,j}$ for “cursor of i’th cfg at j”
- For each $i, j \leqslant |w|$ and $\sigma \in \Sigma$: $\ell_{i,j,\sigma}$ for *“(i, j) cell has σ”*

Yes/no for consistency conditions



- One state + one cursor per row
- one symbol per cell
- First row is initial state + $>w$.
- Last row has accept state

Yes/no for operational conditions



- Each subsequent row is obtained from the preceding by one of the rules of M

BOOL-SAT *is NP-Complete*

- BOOL-SAT is fersibly certified:

  The certificate is the satisfying valuation.

- BOOL-SAT is NP-hard:

  Every NP problem reduces to ND-ONSITE-ACCEPT by padding,

  and ND-ONSITE-ACCEPT $\leqslant$ **BOOL-SAT**.

Normal forms

- Boolean expressions may be arbitrarily complex.

  Can we facilitate eductions by focusing on some that are simple?

- Reductions to ***normal forms*** are all around!

- Decimal fractions (percents):

  3/8 versus 4/11 (.375 vs .364)

- Better: normalized scientific notation for real numbers:

  $123.45 = 1.2345 \times 10^{2}$,

  $0.0012345 = 1.2345 \times 10^{-3}$,

  $1.2345 = 1.2345 \times 10^{0}$

- Display immediately the order of magnitude.

- Polynomials are defined using $+, \times, -$ in any order.

- Putting order in the chaos:

  $\times$ in the scope of $-$, in the scope of $+$.

- $-((x+y)\cdot x)\cdot(1-y) = x^2\cdot y + x\cdot y^2 - x^2 - x\cdot y$

Normal form for boolean expressions

- For boolean expressions: chaos of negations, conjunctions, disjunction
- **Normal form:** negations in scope of conjunctions in scope of disjunctions

$$
\begin{aligned}
-[(x \vee -u) \wedge (y \vee v)] \;=\; & (-x \vee -y) \\
& \wedge (-x \vee -v) \\
& \wedge (u \vee -y) \\
& \wedge (u \vee -v)
\end{aligned}
$$

- **Literals:** variables or their negation.
- **(disjunctive) clauses:** disjunction of literals (1,2,3,0... disjuncts)
- **Conjunctive normal expression (CNF):**

  conjunction of disjunctive clauses

CNF and satisfiability

- More orderly BOOL-SAT: ask only about satisfiability of CNFs:

  CNF-SAT:

  Given a CNF boolean expression E, is it satisfiable?

- We'll show that CNF-SAT is NP-hard.

- NP-hardness of problems would be made easier:

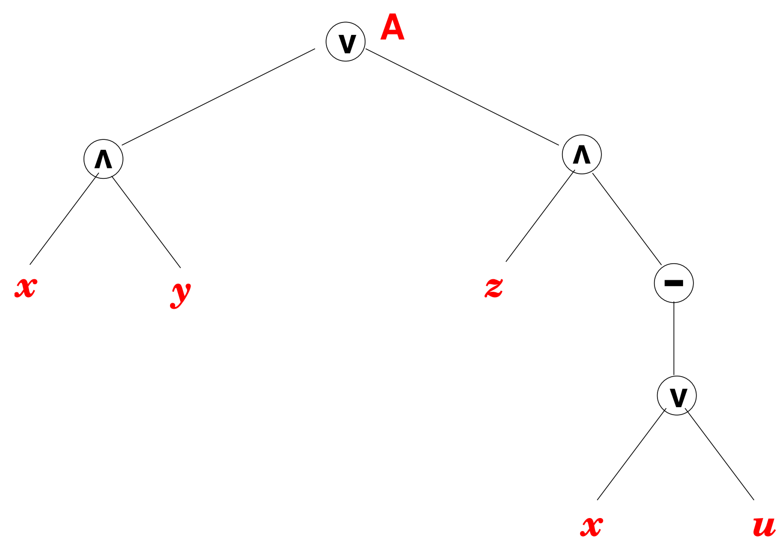
  CNF-SAT $\leqslant_p \mathcal{P}$ easier to show than **BOOL-SAT** $\leqslant_p \mathcal{P}$

CNF-SAT *is NP-hard*

- Method: Reduce bool-sat to cnf-sat.
- Every boolean expression can be converted into an equivalent CNF expression.
- But this does NOT yield the desired reduction!
- Expression E is converted into a CNF equivalent which may be *exponentially longer*!
- However: NO NEED for an equivalent CNF! Suffices a CNF whose ***satisfiability*** is equivalent to the ***satisfiability*** of E.
- We can even restrict attention to ***3CNF*** expressions where each clause has $\leqslant 3$ literals.

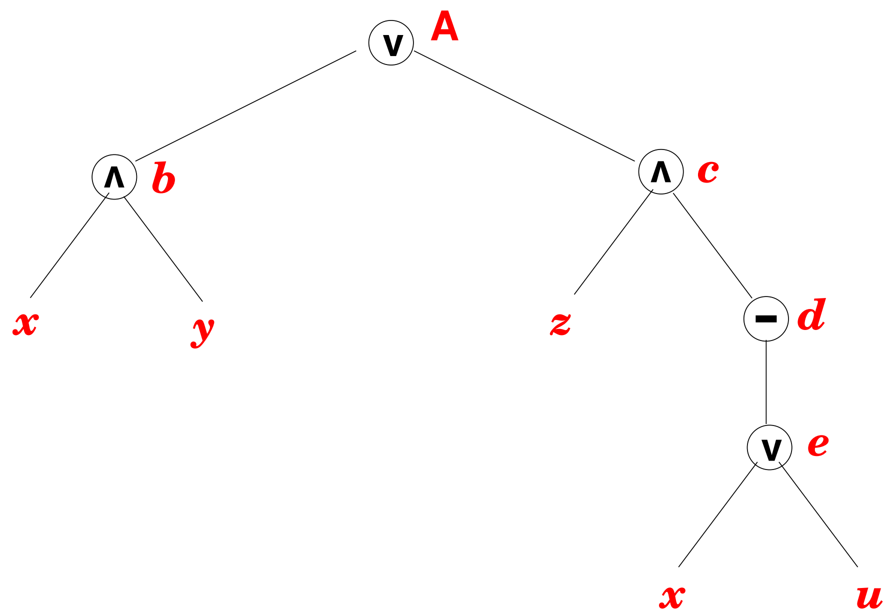
3CNF-Satisfiability

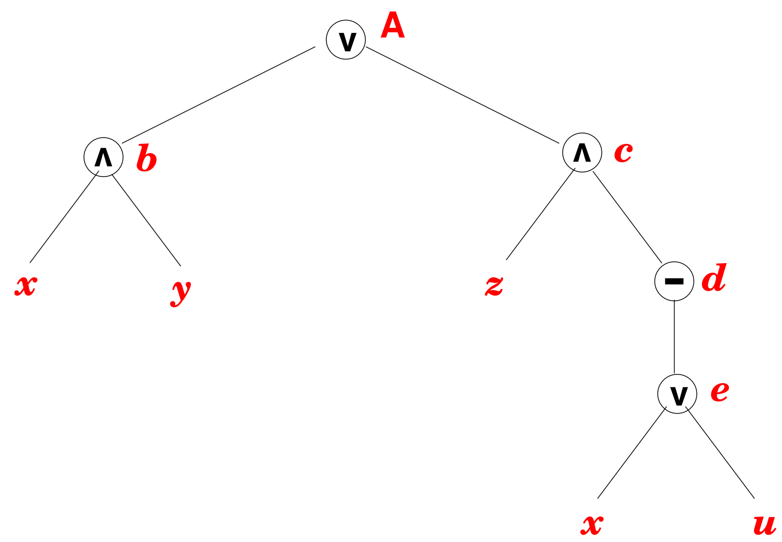
- 3CNF SATISFIABILITY Does a given 3CNF expression have a verifying valuation.
- **BOOL-SAT 3CNF-SAT**
- Example, A is $(x \wedge y) \vee (z \wedge -(x \vee u))$



- Name with fresh variables the compound sub-expressions of A:

$$
\begin{aligned}
a &\equiv A \\
b &\equiv x \wedge y \\
c &\equiv x \wedge u \\
d &\equiv -(x \wedge u) \\
e &\equiv z \vee -(x \wedge u)
\end{aligned}
$$





- Define $A^{=}$ to be the conjunction of

$$(a \leftrightarrow (b \vee c))$$
$$(b \leftrightarrow (x \wedge y))$$
$$(c \leftrightarrow (z \wedge d))$$
$$(d \leftrightarrow -e)$$
$$(e \leftrightarrow (x \vee u))$$

In 3CNF form:

$$(a \leftrightarrow (b \vee c)) \quad \begin{array}{l} \bar{a} \vee b \vee c, \\ a \vee \bar{b}, \\ a \vee \bar{c}, \end{array}$$

$$(b \leftrightarrow (x \wedge y)) \quad \begin{array}{l} \bar{b} \vee x, \\ \bar{b} \vee y \\ \bar{x} \vee \bar{y} \vee b, \end{array}$$

$$(c \leftrightarrow (z \wedge d)) \quad \begin{array}{l} \bar{c} \vee z, \\ \bar{c} \vee d, \\ \bar{z} \vee \bar{d} \vee c, \end{array}$$

$$(d \leftrightarrow -e) \quad \begin{array}{l} \bar{d} \vee \bar{e}, \\ e \vee \bar{d}, \end{array}$$

$$(e \leftrightarrow (x \vee u)) \quad \begin{array}{l} \bar{e} \vee x \vee u, \\ \bar{x} \vee e, \\ \bar{u} \vee e \end{array}$$

- A is satisfiable iff the 3CNF $a \wedge A^{=}$ is satisfiable.
- $a \wedge A^{=}$ is of size linear in the size of A.

Exact-3CNF-Sat

- Further tightening the normal form for boolean expression.
- EXACT-3CNF-SAT:

  Does a given 3CNF expression w/ exactly 3 literals per clause have a satisfying valuation?

- **3CNF-SAT $\leqslant_P$ exact-3cnf-sat**
- Given a 3-CNF A obtain $\rho(A)$ by
  1. Replacing clauses $L_0 \vee L_1$ by

     $(L_0 \vee L_1 \vee y) \wedge (L_0 \vee L_1 \vee \bar{y})$ (y fresh);
  2. Replacing single-literal clauses L by

     $(L \vee y \vee z) \wedge (L \vee y \vee \bar{z}) \wedge (L \vee \bar{y} \vee z) \wedge (L \vee \bar{y} \vee \bar{z})$

NP COMPLETENESS ALL AROUND

INDEP-SET *is NP-complete*

- Define $\rho :$ **EXACT-3CNF** $\leqslant_p$ **INDEP-SET**.

  Try to map exact-3CNF E with k disj-clauses

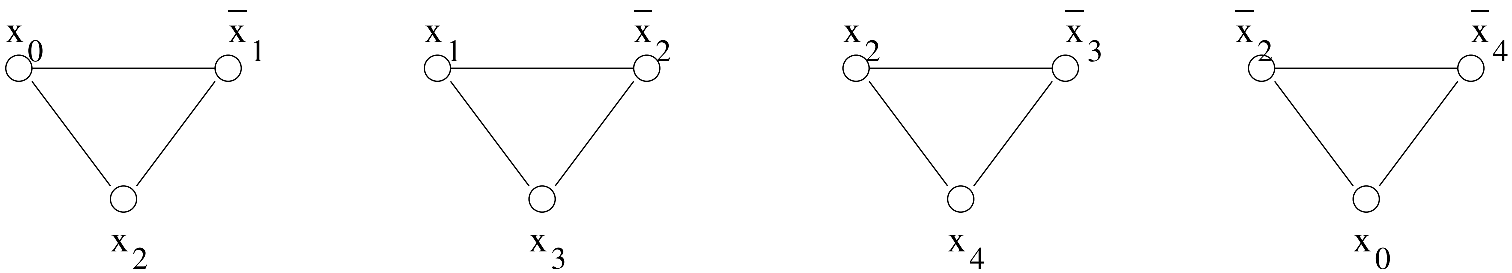
  to graph G + target k.

- First idea: Map each clause to a triangle of literals.

  Satisfying k clauses requires then one vertex per triangle:

$$(x_0 \vee \bar{x}_1 \vee x_2) \ \wedge \ (x_1 \vee \bar{x}_2 \vee x_3) \ \wedge \ (x_2 \vee \bar{x}_3 \vee x_4) \ \wedge \ (\bar{x}_2 \vee \bar{x}_4 \vee x_0)$$

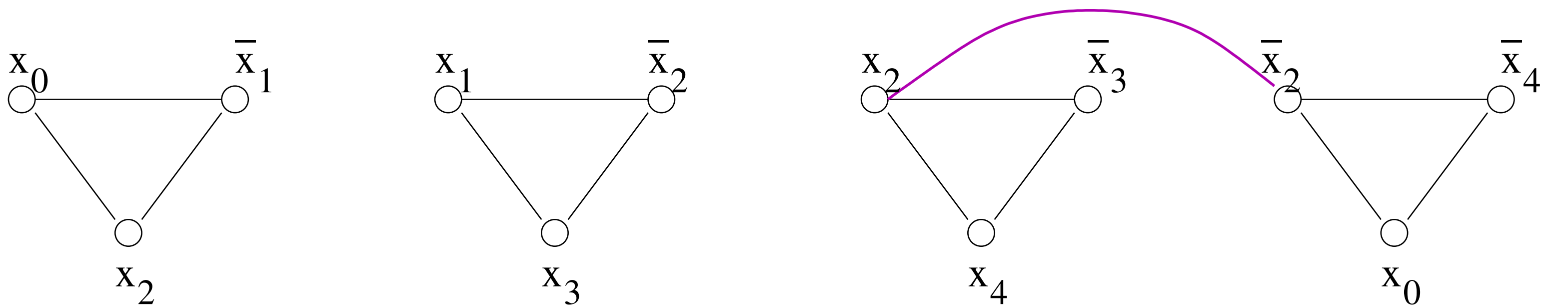
- An initial draft of G :



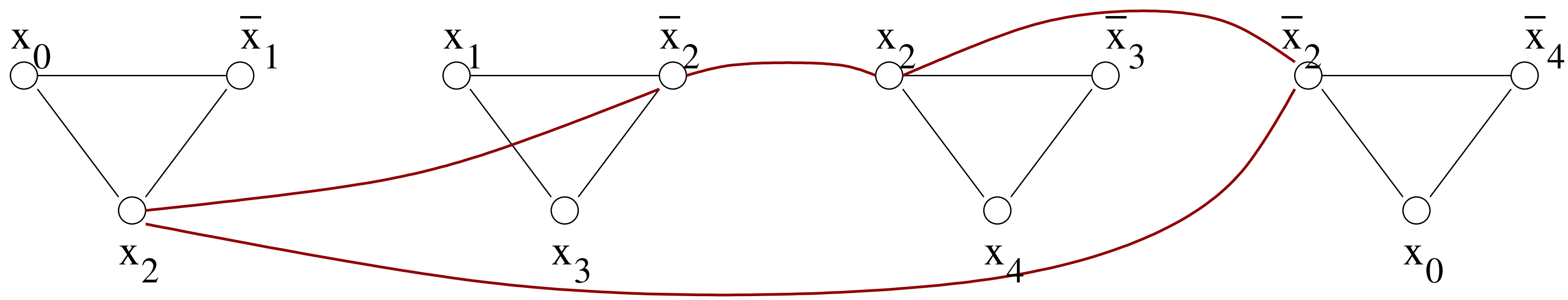
- Choose a vertex in each triangle, eg top left.

  Oops, we are trying to have both x_2 and $\bar{x}_2$ true!

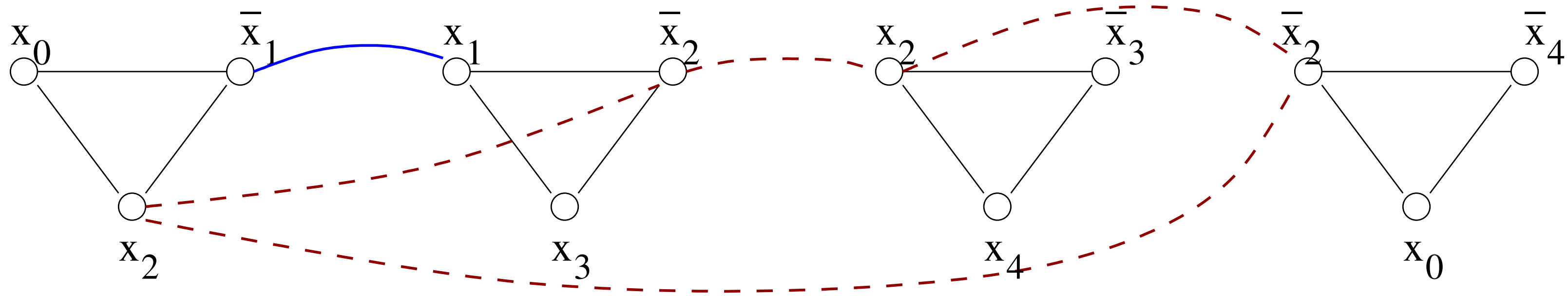
- Consistency edge for x_2:



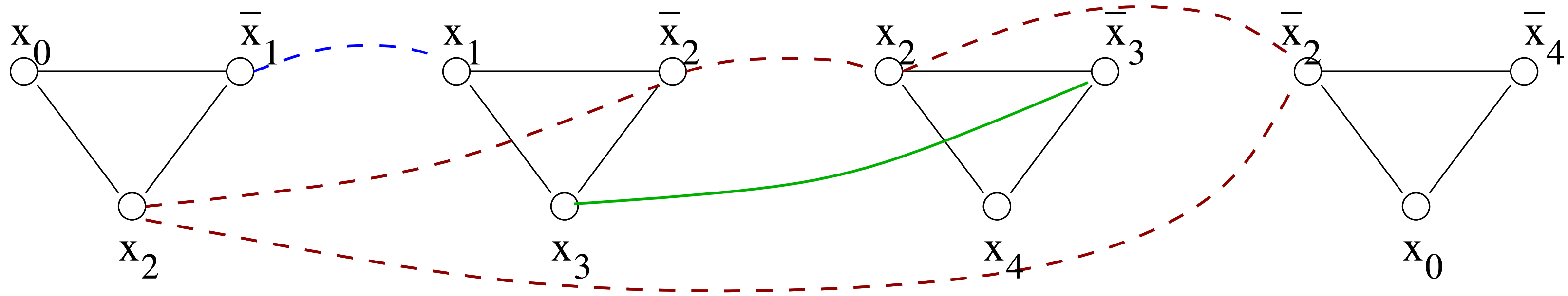
- Additional consistency edges for x_2:



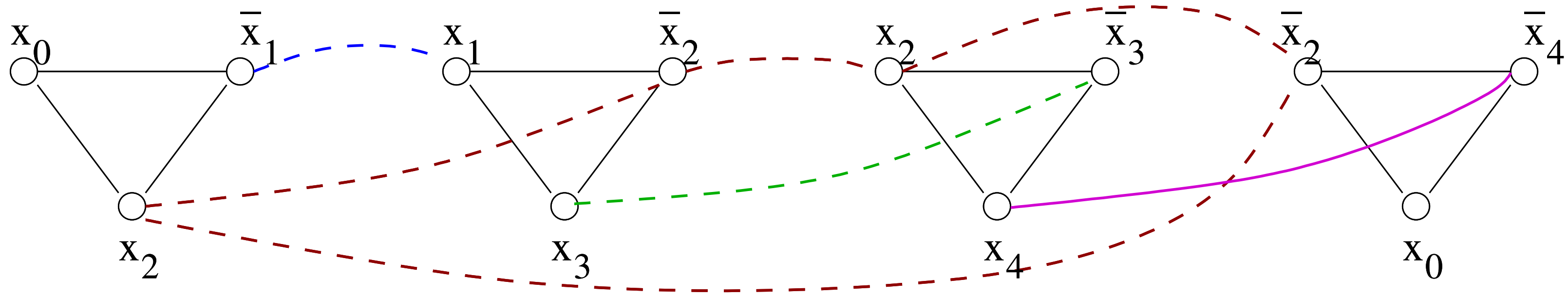
- Consistency edge for x_1:



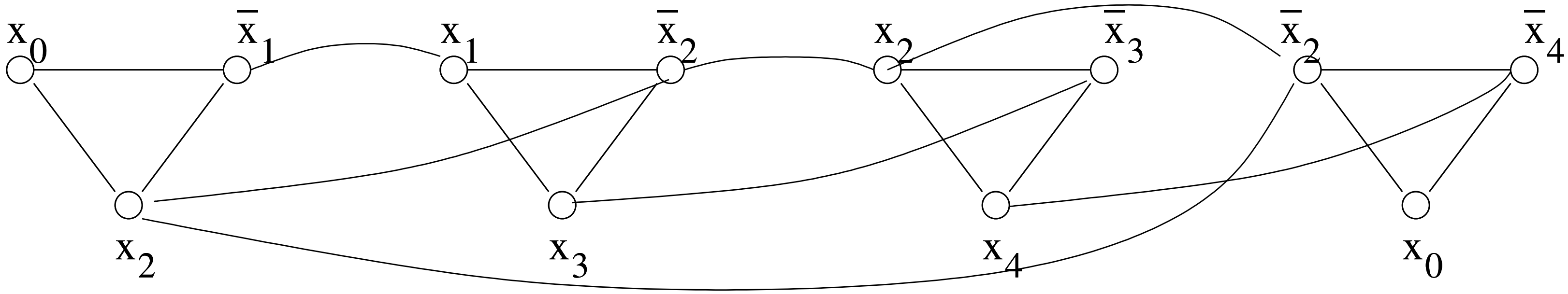
• Consistency edge for x_3:



- Consistency edge for x_4:



• Final graph G:



• If A has a satisfying valuation msV,

then G has an independent-set S of size t,

consisting of vertices true under V.

• If G has an independent set S of size t,

then S must have one vertex per triangle,

and the valuation that verifies the labels of S satisfies A.

Consequence: CLIQUE *is NP-complete*

- We showed that clique is NP,

  and that INDEPENDENT-SET $\leqslant_p$ CLIQUE.

- Since INDEPENDENT-SET is NP-hard, so is CLIQUE.

The INTEGER-PROGRAMMING *problem*

- Dealing with finite sets of linear inequalities,

  such as $2x - 3y + 4z - 7 \geqslant 0$. Quite flexible:

  - Inequality between expressions: $E \geqslant E'$ same as $E - E' \geqslant 0$.
  - Equality: $E = E'$ iff $E \geqslant E'$ and $E' \geqslant E$.
  - Strict inequality: $E > 0$ same as $E - 1 \geqslant 0$.

- INTEGER-PROGRAMMING: Given a set of linear inequalities,

  does it have an integer solution?

- Example: $2x + y \geqslant 0, \ \ -x + 2y - 4 \geqslant 0$ has solution $x = -1, \ y = 2$
- A PTime-certification: The solution is the certificate.
- Snag: Size of solution is non-trivial. Need:

  Exists k s.t. every solvable instance L has a solution of textual size $\leqslant |L|^k$.

INTEGER-PROGRAMMING *is NP-hard*

- CNF-SAT $\leqslant_P$ INTEGER-PROGRAMMING
- Given CNF expression $\quad x \vee y \vee -z \quad - x \vee v \vee z$
- Rephrased as set of inequalities:

$$0 \leqslant x \leqslant 1 \quad 0 \leqslant y \leqslant 1 \quad 0 \leqslant z \leqslant 1 \quad 0 \leqslant v \leqslant 1$$

$$x + y + (1 - z) \geqslant 1 \quad (1 - x) + v + z \geqslant 1$$

- So INTEGER-PROGRAMMING Is NP-hard, and therefore NP-complete.

EXACT-SUM *is NP-complete*

- Recall: Given set S of positive integers and target $t > 0$ is there $P \subseteq S$ such that $\Sigma P = t$.
- Did: it is NP
- We show NP-hardness: $\rho :$ **3cnf-sat** $\leqslant_p$ **exact-sum**

Representing the boolean switch

- Represent a boolean choice by a pair of positive integers:

  The condition $x^+ + x^- = 1$ is equivalent to having one of x^+ and x^- being 1 and the other 0.

- Represent multiple boolean choices by a table of 0/1 digits:

$$
\begin{array}{c|ccc}
N_1^- & 1 & 0 & 0 \\
N_1^+ & 1 & 0 & 0 \\
N_2^- & & 1 & 0 \\
N_2^+ & & 1 & 0 \\
N_3^- & & & 1 \\
N_3^+ & & & 1
\end{array}
$$

- To force each pair to add up to 1 we require that

  the entire table adds up to the decimal $111\ldots1$:

- N_i^+, N_i^- represent a booelan choice for the boolean variable x_i.

Representing clauses

- Extend each row with additional info about the variable,

  in the form of extra digits to the right.

- Say we want ot preprersent the clauses of

  $(x_1 \vee \bar{x}_2 \vee x_3) \;\wedge\; (x_1 \vee x_2 \vee \bar{x}_4) \;\wedge\; (\bar{x}_1 \vee x_2 \vee x_4) \;\wedge\; (x_2 \vee \bar{x}_3 \vee \bar{x}_4)$

- The j'th extra digit to the right indicates whether

  the boolean choice implied by the row makes the j'th clause true:

$$(x_1 \vee \bar{x}_2 \vee x_3) \;\wedge\; (x_1 \vee x_2 \vee \bar{x}_4) \;\wedge\; (\bar{x}_1 \vee x_2 \vee x_4) \;\wedge\; (x_2 \vee \bar{x}_3 \vee \bar{x}_4)$$

					C_1	C_2	C_3	C_4
N_1^+	1	0	0	0	1	1	0	0
N_1^-	1	0	0	0	0	0	1	0
N_2^+	0	1	0	0	0	1	1	1
N_2^-	0	1	0	0	1	0	0	0
N_3^+	0	0	1	0	1	0	0	0
N_3^-	0	0	1	0	0	0	0	1
N_4^+	0	0	0	1	0	0	1	0
N_4^-	0	0	0	1	0	1	0	1
	1	1	1	1	1^+	1^+	1^+	1^+

Balancing the sums

					C_1	C_2	C_3	C_4
N_1^+	1	0	0	0	1	1	0	0
N_1^-	1	0	0	0	0	0	1	0
N_2^+	0	1	0	0	0	1	1	1
N_2^-	0	1	0	0	1	0	0	0
N_3^+	0	0	1	0	1	0	0	0
N_3^-	0	0	1	0	0	0	0	1
N_4^+	0	0	0	1	0	0	1	0
N_4^-	0	0	0	1	0	1	0	1
	0	0	0	0	2	0	0	0
	0	0	0	0	1	0	0	0
	0	0	0	0	0	2	0	0
	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	2	0
	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	2
	0	0	0	0	0	0	0	1
$t =$	1	1	1	1	4	4	4	4

- The given boolean expression is satisfiable iff

  the 16 numbers above, 10,001,100 . . . 10,2,1,

  can be added to 11,114,444.

- EXACT-SUM is NP-complete,

- Why not “fill-up” with 1,1, adding up to 3,

  in place of 1,2, adding up to 4? (Answer: We want distinct integers)

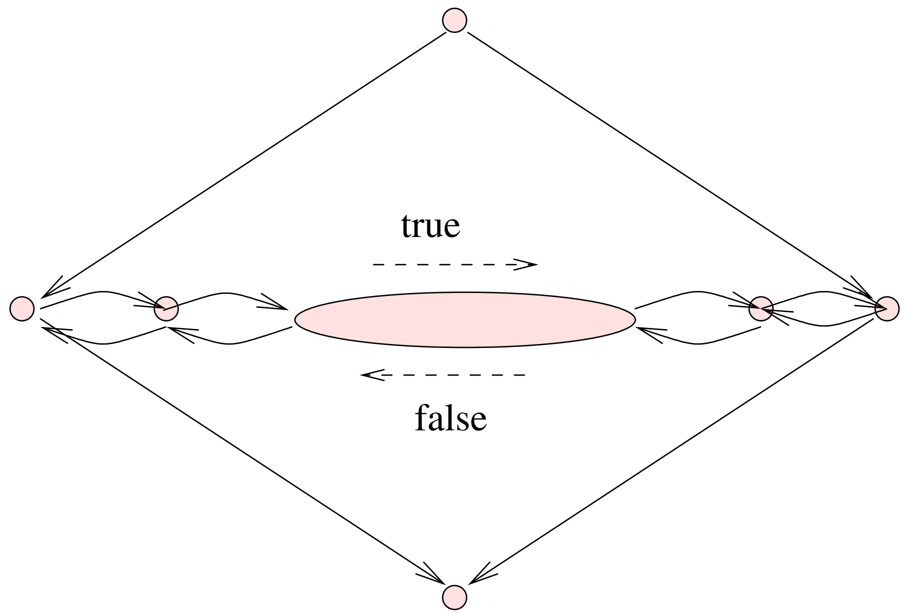
*HAMILTONIAN-PATH

- HAMILTONIAN-PATH: Given a directed graph $G = (V, E)$,
  does it have a path visiting each vertex exactly once
- The problem has a feasible certification: the certificate is the path.

* *The truth gadget for* **HAMILTONIAN-PATH**

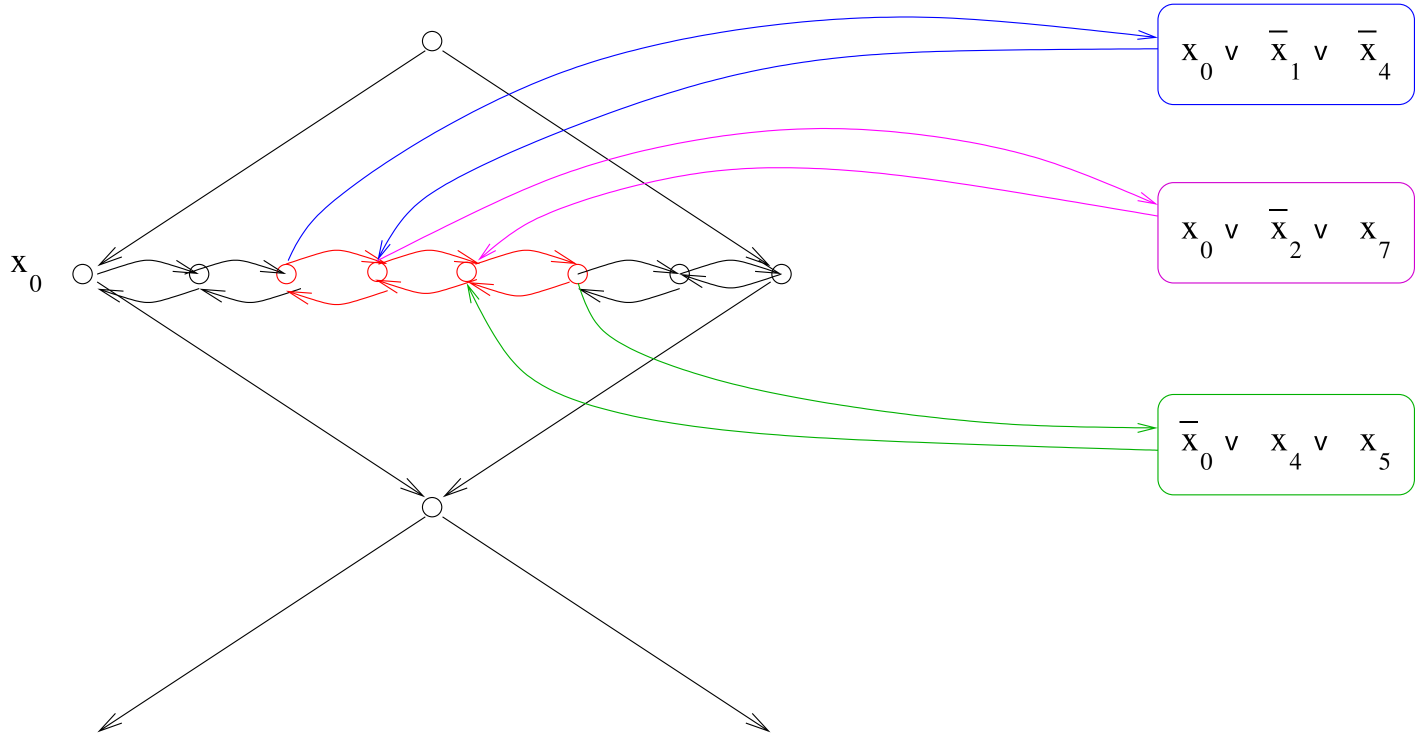
- 3CNF-SAT $\leqslant_P$ HAMILTONIAN-PATH

- A boolean gadget:



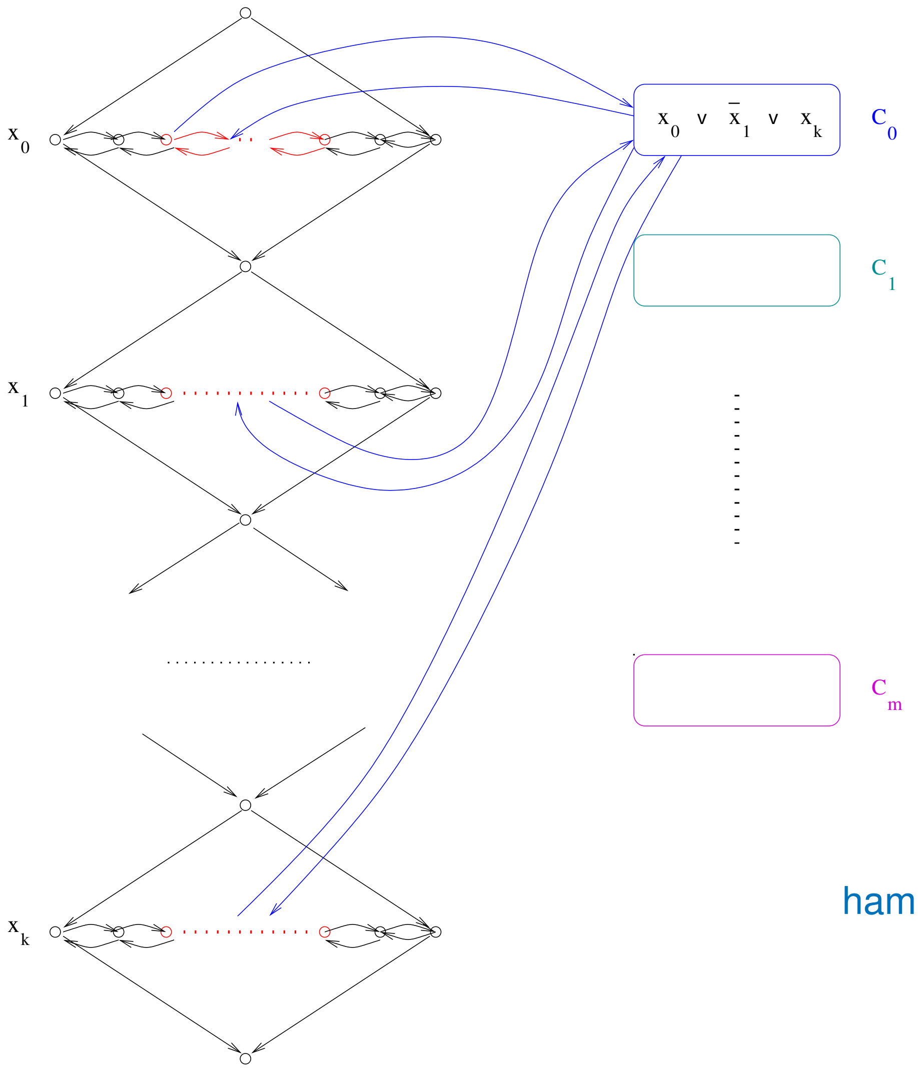
⋆ *The gadget used as a switchboard*

A boolean switchboard:



The x_0 switchboard used positively by two clauses and negatively by one

Combining the switchboards



hamiltonian-path is NP-complete.