TIME COMPLEXITY

Measuring computational complexity

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

  = number of cfgs in computation trace
- Steps on ***Turing machines***: They count moves honestly.

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 Turing machine M over Σ

  let $T_M(w)$ be the number of cfg's in the

  trace of M for input $w \in \Sigma^*$, if defined.
- Given a function $f : \mathbb{N} \to \mathbb{N}$), M $\boxed{\textit{runs within time } f}$

  if $T_M(w) \leqslant f(|w|)$ for all inputs w.
- Example: if M runs within time $n \mapsto n^2$

  then $T_M(\mathbf{abcde}) \leqslant 25$.
- Note that if M runs within time f and $f \leqslant g$

  then M runs within within g as well.

Which machine model

- Why Turing machines are the reference?

  Because they don't cheat.

- But perhaps they are too simple.

- E.g. to compute $w \mapsto w \cdot w$

  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 $< 6\,|w|$ steps, for some small constant c.

Comparing asymptotic behaviors

- By ***asymptotic behavior*** of a function $f : \mathbb{N} \to \mathbb{N}$ we mean its behavior for all sufficiently large input.
- Examples: Asymptotically, $n^3 > 100n^2$ (for $n > 100$) and $100\,n^3 < 2^n$ (for $n > 15$).

Comparing asymptotic behaviors

- By ***asymptotic behavior*** of a function $f : \mathbb{N} \to \mathbb{N}$
  we mean its behavior for all sufficiently large input.
- Examples: Asymptotically, $n^3 > 100n^2$ (for $n > 100$)
  and $100\,n^3 < 2^n$ (for $n > 15$).
- In Geometry, an *asymptote* of a curve
  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.

Coefficients ignored: big-O notation

- Implementation choices, such as hardware or size of alphabet,

  are important in determining performance,

  but we wish to abstract away from them, to obtain a broader vision.

- A function g is $\boxed{\textit{of order}}$ f if there are $c, k > 0$ s.t.

  $g(n) \leqslant c \cdot f(n)$ for all $n \geqslant k$.

- We say then that g is $O(f)$ ("big-O of f").

Coefficients ignored: big-O notation

- Implementation choices, such as hardware or size of alphabet,

  are important in determining performance,

  but we wish to abstract away from them, to obtain a broader vision.

- A function g is $\boxed{\textit{of order}}$ f if there are $c, k > 0$ s.t.

  $g(n) \leqslant c \cdot f(n)$ for all $n \geqslant k$.

- We say then that g is $O(f)$ ("big-O of f").

- Convention:

  Use n as a catch-all variable for natural numbers,

  writing eg $O(n^2)$ for $O(n \mapsto n^2)$,

  that is "$O(f)$ where $f(n) = n^2$."

Time complexity classes

- TM M is $\boxed{\textit{in time } f}$ if T_M is $O(f)$.
- We write **Time**(f) for the collection of languages recognized by a Turing acceptor in time $O(f)$.
- The f's of interest are non-decreasing:
  $f(n+1) \geqslant f(n)$ for all n.
- Examples: $\log n,\ n,\ n\log n,\ n^2,\ n^5,\ 2^n,\ 2^{n^2},\ n!,\ n^n$.
- Similar notation for transducers.

The Time Hierarchy Theorem

- We can expect that significantly more computation time implies that more functions are computable.
- This is mostly true:
- **Time Hierarchy Theorem.** *Assume*
  - $t, T : \mathbb{N} \to \mathbb{N}$ are "reasonable"; and
  - $\dfrac{t(n) \cdot \log(t(n))}{T(n)} \quad \to \quad 0$ as $n \to \infty$

  Then there is a language recognized in $\textbf{Time}(t)$ *but not in* $\textbf{Time}(T)u$.
- Alternative phrasing: $t(n) \cdot \log(t(n)) \ = o(T(n))$ ("little o").

Instances of the Time-Hierarchy Theorem

- $\mathsf{Time}(n) \subsetneq \mathsf{Time}(n^2) \subsetneq \mathsf{Time}(n^3)$
$\subsetneq \mathsf{Time}(n^{3.001}) \subsetneq \mathsf{Time}(2^n) \subsetneq \mathsf{Time}(3^n)$
$\subsetneq \mathsf{Time}(2^{n^2}) \subsetneq \mathsf{Time}(n!) \subsetneq \mathsf{Time}(n^n)$

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:

  $a^n = b^{cn}$ where $c = \log_b(a)p = \log b / \log a$,
- But polynomial and exponential growth-rates tell very different stories:

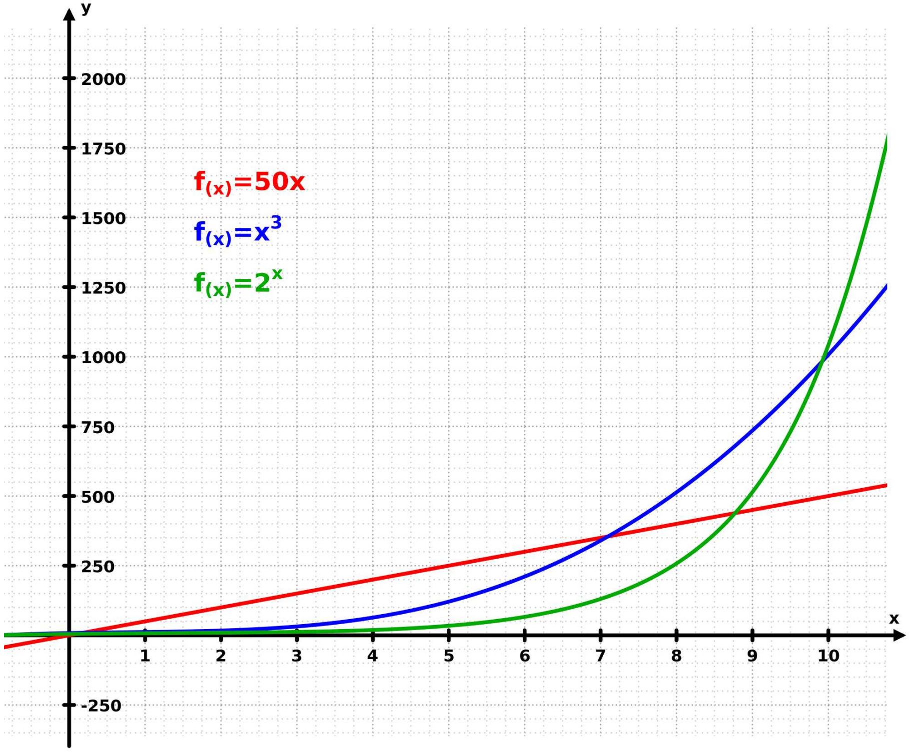
  If an algorithm runs 2^n steps on input of size n, then

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

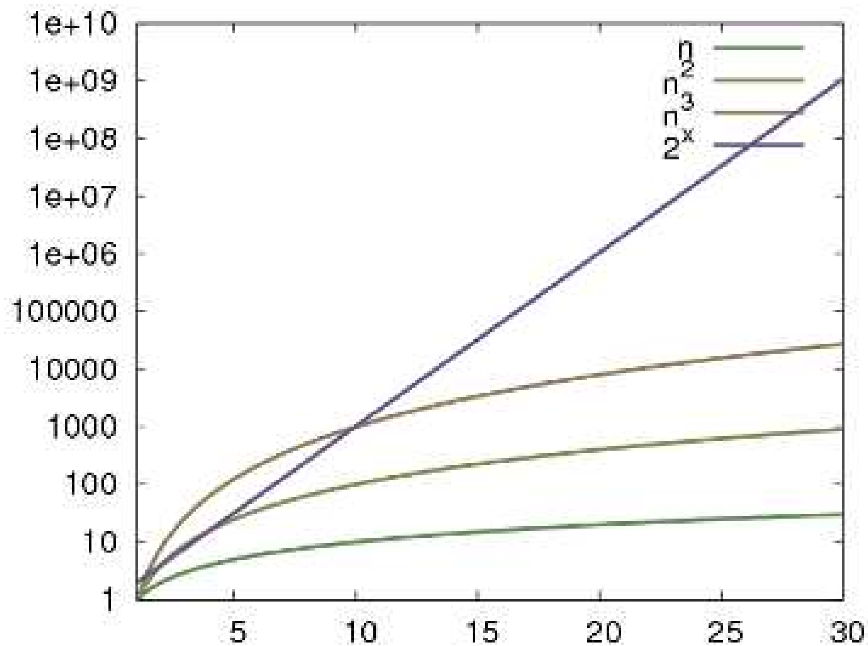
  It is believed that there are $10^{90} \approx 2^{300}$ quarks in the universe.

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: 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.
- We can therefore consider informal algorithms without worrying about low level implementation.

  Exception: linear & quasi-linear ($n \log n$).

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{\, w \mid w \cdot w \in L \,\}$ PTime-decidable?

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{\, w \mid w \cdot w \in L \,\}$ PTime-decidable?

  Yes.

  Deciding $w \cdot w \in L$ takes time $\leqslant c\,|w \cdot w|^k = 2^k \cdot |w|^k$,

  so L' is decidable in time $O(n^k)$ too.

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- What if we upgrade to $w \cdot w \cdot w$?

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- What if we upgrade to $w \cdot w \cdot w$?

  Upgrade 2^k to 3^k, Still **Time**(n^k).

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{\, w \mid w \cdot w \in L \,\}$ PTime-decidable?
- What about $L'' = \{\, w \mid |w|\, w \in L \,\}$?

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time $c \cdot n^k$.

- Is $L' = \{\, w \mid w \cdot w \in L \,\}$ PTime-decidable?
- What about $L'' = \{\, w \mid |w|\, w \in L \,\}$?

  $||w|\, w| = |w \cdot \cdots \cdot w| = |w|^2$.

  Deciding $|w|\, w \in L$ takes time $\leqslant (|w|^2)^k = |w|^{2k}$

  Still PTime!

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time n^k (glossing over the big-O.)

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- Is $L^2 = L \cdot L$ Ptime decidable?

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time n^k (glossing over the big-O.)

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- Is $L^2 = L \cdot L$ Ptime decidable?

  On input x cycle through all splits $x = u \cdot v$.

  How many splits are there?

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time n^k (glossing over the big-O.)

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- Is $L^2 = L \cdot L$ Ptime decidable?

  On input x cycle through all splits $x = u \cdot v$.

  How many splits are there?

  Take $x = \mathbf{abcde}$. Count the positions of the dot:

$$x = \varepsilon \cdot \mathbf{abcde},\ \mathbf{a} \cdot \mathbf{bcde},\ \mathbf{ab} \cdot \mathbf{cde},\ \mathbf{abc} \cdot \mathbf{de},\ \mathbf{abcd} \cdot \mathbf{e}$$

  There are $|x|$ splits.

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time n^k (glossing over the big-O.)

- Is $L' = \{\, w \mid w \cdot w \in L \,\}$ PTime-decidable?
- Is $L^2 = L \cdot L$ Ptime decidable?

  On input x cycle through all splits $x = u \cdot v$.

  How many splits are there?

  Each split $x = u \cdot v$ takes $\leqslant |u|^k + |v|^k < 2|x|^k$ steps.

  So L^2 is decidable in time $O(n^{k+1})$.

Examples

Suppose $L \subseteq \Sigma^*$ is PTime decidable,

say within time n^k (glossing over the big-O.)

- Is $L' = \{ w \mid w \cdot w \in L \}$ PTime-decidable?
- Is $L^2 = L \cdot L$ Ptime decidable?

  On input x cycle through all splits $x = u \cdot v$.

  How many splits are there?

  Each split $x = u \cdot v$ takes $\leqslant |u|^k + |v|^k < 2|x|^k$ steps.

  So L^2 is decidable in time $O(n^{k+1})$.
- ***Note:*** We have not attempted to refine the bounds.

  The order-of-magnitude trounces such concerns.

⋆*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, the reference to "algorithms" is justified.

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

SHOWING PTIME DECIDABILITY

Major PTime decision-problems

- **CONNECTIVITY**:

  Given a graph $G = (V, E)$, *is it connected?* (Dijkstra, 1969)

Major PTime decision-problems

- **CONNECTIVITY**:

  Given a graph $G = (V, E)$, *is it connected?* (Dijkstra, 1969)

- **LINEAR-INEQUAL**:

  Given a set of linear inequalities, is there a solution with real numbers? (Khachian, 1979)

  Example: $3x + y \geqslant 0, \;\; x + 3y \leqslant 0$

Major PTime decision-problems

- **CONNECTIVITY**:

  Given a graph $G = (V, E)$, *is it connected?* (Dijkstra, 1969)

- **LINEAR-INEQUAL**:

  Given a set of linear inequalities, is there a solution with real numbers? (Khachian, 1979)

  Example: $3x + y \geqslant 0, \;\; x + 3y \leqslant 0$

- **PRIMALITY**:

  Given a natural number, is it prime (Agrawal, Kayal & Saxena, 2006)

Memoization: caching data for repeated use

- ***Memoization*** = memorize information for future use
  (Greek: *mnémé* = memory).

  Also called ***dynamic programming*** algorithm,
  because information is cached "dynamically" over times.

Memoization: caching data for repeated use

- ***Memoization*** = memorize information for future use

  (Greek: *mnémé* = memory).

  Also called ***dynamic programming*** algorithm,

  because information is cached "dynamically" over times.

- The CYK algorithm is an example. Here's a related one.

  Theorem. *If $L \subseteq \Sigma^*$ is PTime-decidable then so is L^**

Memoization: caching data for repeated use

- ***Memoization*** = memorize information for future use
  (Greek: *mnémé* = memory).
  Also called ***dynamic programming*** algorithm,
  because information is cached "dynamically" over times.
- The CYK algorithm is an example. Here's a related one.
  Theorem. *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.

Memoization: caching data for repeated use

- ***Memoization*** = memorize information for future use
  (Greek: *mnémé* = memory).
  Also called ***dynamic programming*** algorithm,
  because information is cached "dynamically" over times.
- The CYK algorithm is an example. Here's a related one.

  Theorem. *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!!

Memoization: caching data for repeated use

- ***Memoization*** = memorize information for future use
  (Greek: *mnémé* = memory).
  Also called ***dynamic programming*** algorithm,
  because information is cached "dynamically" over times.
- The CYK algorithm is an example. Here's a related one.
  Theorem. *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! So...?

PTime is closed under star

Theorem. *If $L \subseteq \Sigma^*$ is PTime-decidable then so is L^**

- For input ε the answer is "yes".

PTime is closed under star

Theorem. *If $L \subseteq \Sigma^*$ is PTime-decidable then so is L^**

- For input ε the answer is "yes".
- We generate the set S of substrings of $w = \sigma_1 \cdots \sigma_n$ that are in L^*

PTime is closed under star

Theorem. *If $L \subseteq \Sigma^*$ is PTime-decidable then so is L^**

- For input ε the answer is "yes".
- We generate the set S of substrings of $w = \sigma_1 \cdots \sigma_n$ that are in L^*
- Calculate for successive i the i-long elements of S:

  $S_i = \{\, \sigma_{m+1} \cdots \sigma_{m+i} \in L^+ \mid 0 \leqslant m \leqslant n-i \,\}$

PTime is closed under star

Theorem. *If $L \subseteq \Sigma^*$ is PTime-decidable then so is L^**

- For input ε the answer is "yes".
- We generate the set S of substrings of $w = \sigma_1 \cdots \sigma_n$ that are in L^*
- Calculate for successive i the i-long elements of S:

  $S_i = \{\ \sigma_{m+1} \cdots \sigma_{m+i} \in L^+ \ \mid\ 0 \leqslant m \leqslant n-i\ \}$
- $S_1 = \{\sigma_i \mid \sigma_i \in L\}$.

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

  w's substrings x of length i start at σ_j for $j \in [1..n-i)$

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

  w 's substrings x of length i start at σ_j for $j \in [1..n-i)$

  For each such x consider the $i{-}1$ possible non-trivial splits

  $x = y \cdot z$, say $|y| = d \in (0..i)$.

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

  w 's substrings x of length i start at σ_j for $j \in [1..n-i)$

  For each such x consider the $i{-}1$ possible non-trivial splits

  $x = y \cdot z$, say $|y| = d \in (0..i)$.

  If $y \in S_d$ and $z = S_{i-d}$ add x to S_i.

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

  w's substrings x of length i start at σ_j for $j \in [1..n-i)$

  For each such x consider the $i-1$ possible non-trivial splits

  $x = y \cdot z$, say $|y| = d \in (0..i)$.

  If $y \in S_d$ and $z = S_{i-d}$ add x to S_i.

- Finally w is accepted iff $w \in S_n$.

Computing S_i *for* $i \in [2..n]$

- Initially S_i is empty.

  w 's substrings x of length i start at σ_j for $j \in [1..n-i)$

  For each such x consider the $i-1$ possible non-trivial splits

  $x = y \cdot z$, say $|y| = d \in (0..i)$.

  If $y \in S_d$ and $z = S_{i-d}$ add x to S_i.

- Finally w is accepted iff $w \in S_n$.

- The algorithm has three nested loops,

  each iterating $\leqslant |w|$ times,

  so running time is $O(n^3)$.

PTIME CERTIFICATION

Reminder: Certifications

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

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

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

$$w \text{ satisfies } \mathcal{P} \quad \text{IFF} \quad c \vdash w \text{ for some certificate } c$$

- We showed that a problem $\mathcal{P}$ is SD

  iff it has a decidable certification.

Feasible-certification

- A certification $\vdash$ for $\mathcal{P}$ is $\boxed{\textit{PTime}}$

  if $c \vdash w$ is PTime in $|w|$ ***(only!)***. We write then $c \vdash_{\mathcal{P}} w$.

- 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.

- An equivalent definition: A certification $\vdash$ for $\mathcal{P}$ is $\boxed{\textit{PTime}}$

  if $c \vdash w$ is PTime, and $|c| \leqslant |w|^k$ for some k.

Examples: Scheduling problems

- Scheduling problems: Can we fit stuff within given constraints.
- INTEGER-PARTITION: *Given a set S of positive integers with an even total sum*

  is there a set $P \subseteq S$ such that $\Sigma P = \Sigma(S - P)$?
- Certificate for S: P.

  Certification is PTime: Checking $P \subseteq S$ and $\Sigma P = \Sigma(S - P)$.

  P-size: $|P| \leqslant |S|$
- EXACT-SUM *Given set S of positive integers, and target $t > 0$,*

  is there $P \subseteq S$ such that $\Sigma P = t$?
- Certificate for S, t: The subset P.

  P-Time: Check $P \subseteq S$ and $\Sigma P = t$.

  P-size: $|P| \leqslant |S|$.

Examples: Solvability problems

- Recall INTEGER-EQUATION:

  Given integer polynomial P,

  is there an integer solution?

- That problem is undecidable.

  But consider limiting textual size of the solution:

  - BOUNDED-INTEGER-EQUATION:

    Given integer polynomial P and a bound b,

    is there a solution of textual size $\leqslant |b|$?

- Certificate: a solution.

- Certification relation: V solves E.

- This is doable in time $O(n^5)$.

PTIME REDUCTIONS

Recall: Computable reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \to \mathcal{Q} - \textbf{instncs}$ is computable

  we write $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$.

Recall: Computable reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \to \mathcal{Q} - \textbf{instncs}$ is computable

  we write $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$.

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

  then so is $\mathcal{P}$.

Recall: Computable reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \to \mathcal{Q} - \textbf{instncs}$ is computable

  we write $\rho : \mathcal{P} \leqslant_c \mathcal{Q}$.

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

  then so is $\mathcal{P}$.

- Easy exercise: $\mathcal{P}$ is decidable iff $\mathcal{P} \leqslant_c \{0, 1\}$.

PTime reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \rightarrow \mathcal{Q} - \textbf{instncs}$ is PTime

  we write $\rho : \mathcal{P} \leqslant_{p} \mathcal{Q}$.

PTime reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \to \mathcal{Q} - \textbf{instncs}$ is PTime

  we write $\rho : \mathcal{P} \leqslant_{p} \mathcal{Q}$.

- If $\mathcal{P} \leqslant_{p} \mathcal{Q}$ and $\mathcal{Q}$ is PTime,

  then so is $\mathcal{P}$.

PTime reductions

- When a reduction $\rho : \mathcal{P} - \textbf{instncs} \rightarrow \mathcal{Q} - \textbf{instncs}$ is PTime

  we write $\rho : \mathcal{P} \leqslant_{p} \mathcal{Q}$.

- If $\mathcal{P} \leqslant_p \mathcal{Q}$ and $\mathcal{Q}$ is PTime,

  then so is $\mathcal{P}$.

- Easy exercise: $\mathcal{P}$ is PTime iff $\mathcal{P} \leqslant_p \{0,1\}$.

Composition of PTime functions is PTime

- **Theorem.**

  PTime is closed under composition:

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

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

Composition of PTime functions is PTime

- **Theorem.**

  PTime is closed under composition:

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

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

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

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

Composition of PTime functions is PTime

- **Theorem.**

  PTime is closed under composition:

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

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

- 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 $\leqslant c \cdot |w|^k$ steps,

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

Composition of PTime functions is PTime

- **Theorem.**

  PTime is closed under composition:

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

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

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

Composing PTime-reductions

- Since the composition of computable functions is again computable, we had:

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

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

Composing PTime-reductions

- Since the composition of computable functions is again computable, we had:

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

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

- But PTime is also closed under composition,

  so we similarly have:

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

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

Composing PTime-reductions

- Since the composition of computable functions is again computable, we had:

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

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

- But PTime is also closed under composition,

  so we similarly have:

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

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

- A prize for amalgamating the polynomial run-times:

  For example, reducibility in quadratic time

  is ***not*** closed under composition!

EXAMPLES

The disjoint sum

- Fix an alphabet Σ and a fresh Symbol, say $@$.

  For $L, K \subseteq \Sigma^*$ define

  $L \oplus K =_{\mathrm{df}} L \cup @K$ ($@K$ is $\{@\} \cdot K$).

- So if $w \in L \cap K$ then

  w and $@w$ are distinct elements of $L \oplus K$.

The disjoint sum

- Fix an alphabet Σ and a fresh Symbol, say $@$.

  For $L, K \subseteq \Sigma^*$ define

  $L \oplus K =_{\mathrm{df}} L \cup @K$ ($@K$ is $\{@\} \cdot K$).

- So if $w \in L \cap K$ then

  w and $@w$ are distinct elements of $L \oplus K$.

- Example: L = towns in IN, K = towns in NY.

  `Bloomington` is an IN string, `@Bloomington` is a NY string.

The disjoint sum

- Fix an alphabet Σ and a fresh Symbol, say $@$.

  For $L, K \subseteq \Sigma^*$ define

  $L \oplus K =_{\mathrm{df}} L \cup @K$ ($@K$ is $\{@\} \cdot K$).

- So if $w \in L \cap K$ then

  w and $@w$ are distinct elements of $L \oplus K$.

- Example: L = towns in IN, K = towns in NY.

  `Bloomington` is an IN string, `@Bloomington` is a NY string.

- For finite L, K the size of $L \oplus K$

  is the sum of the size of L and the size of K.

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?

  $A \cup @B \cup @@C$

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.

  ρ is the identity on L.

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.
- Define a reduction $\rho : K \leqslant_p L \oplus K$.

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.
- Define a reduction $\rho : K \leqslant_p L \oplus K$.

  $\rho(w) = @w$

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.
- Define a reduction $\rho : K \leqslant_p L \oplus K$.
- Define a reduction $\rho : L \oplus K \leqslant_p K \oplus L$.

Riddles about $\oplus$

- What is $A \oplus (B \oplus C)$?
- Define a reduction $\rho : L \leqslant_p L \oplus K$.
- Define a reduction $\rho : K \leqslant_p L \oplus K$.
- Define a reduction $\rho : L \oplus K \leqslant_p K \oplus L$.

  $\rho(w) =$ if $hd(w) = @$ then $tl(w)$ else $@w$.

Integer-Partition reduces to Exact-Sum

- INTEGER-PARTITION:

Given $S \subseteq \mathbb{N}$, *is there* $P \subset S$ *s.t.* $\Sigma P = \Sigma S - P$

(i.e. both are $\frac{1}{2}(\Sigma S)$

- EXACT-SUM:

Given $S \subseteq \mathbb{N}$ *and a target value* $t \in \mathbb{N}$,

is there $P \subset S$ *s.t.* $\Sigma P = t$.

Integer-Partition reduces to Exact-Sum

- $\textbf{INTEGER-PARTITION}$:

  Given $S \subseteq \mathbb{N}$, *is there* $P \subset S$ *s.t.* $\Sigma P = \Sigma S - P$

  (i.e. both are $\frac{1}{2}(\Sigma S)$

- $\textbf{EXACT-SUM}$:

  Given $S \subseteq \mathbb{N}$ *and a target value* $t \in \mathbb{N}$,

  is there $P \subset S$ *s.t.* $\Sigma P = t$.

- Define $\rho :$ $\textbf{INTEGER-PARTITION} \leqslant_p \textbf{EXACT-SUM}$.

Integer-Partition reduces to Exact-Sum

- INTEGER-PARTITION:

Given $S \subseteq \mathbb{N}$*, is there* $P \subset S$ *s.t.* $\Sigma P = \Sigma S - P$

(i.e. both are $\frac{1}{2}(\Sigma S)$

- EXACT-SUM:

Given $S \subseteq \mathbb{N}$ *and a target value* $t \in \mathbb{N}$ *,*

is there $P \subset S$ *s.t.* $\Sigma P = t$*.*

- Define $\rho:$ INTEGER-PARTITION $\leqslant_p$ EXACT-SUM.
- If ΣS is even,

  let $\rho(S)$ be the instance $(S, \Sigma S/2)$ of EXACT-SUM.

Integer-Partition reduces to Exact-Sum

- **INTEGER-PARTITION**:

  Given $S \subseteq \mathbb{N}$, *is there* $P \subset S$ *s.t.* $\Sigma P = \Sigma S - P$

  (i.e. both are $\frac{1}{2}(\Sigma S)$

- **EXACT-SUM**:

  Given $S \subseteq \mathbb{N}$ *and a target value* $t \in \mathbb{N}$ *,*

  is there $P \subset S$ *s.t.* $\Sigma P = t$.

- Define $\rho :$ **INTEGER-PARTITION** $\leqslant_p$ **EXACT-SUM**.
- If ΣS is even,

  let $\rho(S)$ be the instance $(S, \Sigma S/2)$ of **EXACT-SUM**.
- If ΣS is odd,

  let $\rho(S)$ be some some "no" instance of **EXACT-SUM**,

  say $(\{1\}, 2)$.

Integer-Partition reduces to Exact-Sum

- INTEGER-PARTITION:

Given $S \subseteq \mathbb{N}$, *is there* $P \subset S$ *s.t.* $\Sigma P = \Sigma S - P$

(i.e. both are $\frac{1}{2}(\Sigma S)$

- EXACT-SUM:

Given $S \subseteq \mathbb{N}$ *and a target value* $t \in \mathbb{N}$ *,*

is there $P \subset S$ *s.t.* $\Sigma P = t$.

- Define $\rho :$ INTEGER-PARTITION $\leqslant_p$ EXACT-SUM.
- If ΣS is even,

  let $\rho(S)$ be the instance $(S, \Sigma S/2)$ of EXACT-SUM.
- If ΣS is odd,

  let $\rho(S)$ be some some "no" instance of EXACT-SUM,

  say $(\{1\}, 2)$.
- ρ is in PTime, and is a reduction!

Exact-Sum reduces to Integer-Partition

- **INTEGER-PARTITION** is a special case of **EXACT-SUM**,

  so it was easy to define a reduction in this order.

- Surprisingly, we also have the converse:

  Define $\rho : \textbf{I-P} \leqslant_p \textbf{E-S}$

Exact-Sum reduces to Integer-Partition

- INTEGER-PARTITION is a special case of EXACT-SUM,

  so it was easy to define a reduction in this order.

- Surprisingly, we also have the converse:

  Define $\rho :$ I-P $\leqslant_p$ E-S

- Given instance (S, t) of E-S let $n = \Sigma S$.

  Note: $t < n$, otherwise (S, t) is trivially not in E-S.

Exact-Sum reduces to Integer-Partition

- **INTEGER-PARTITION** is a special case of **EXACT-SUM**,

  so it was easy to define a reduction in this order.

- Surprisingly, we also have the converse:

  Define $\rho :$ **I-P** $\leqslant_p$ **E-S**

- Given instance (S, t) of **E-S** let $n = \Sigma S$.

  Note: $t < n$, otherwise (S, t) is trivially not in **E-S**.

- Idea: augment S with the number $2n - t$, which is $> n$,

  and with $n{+}t$ which totals the result to $4n$.

The reduction

- Let ρ map (S, t) to $S' =_{\text{df}} S \cup \{n + t, 2n - t\}$.

  So S' adds up to $4n$.

The reduction

- Let ρ map (S, t) to $S' =_{\mathrm{df}} S \cup \{n+t,\, 2n-t\}$.
  So S' adds up to $4n$.
- Claim: There is a $P \subseteq S$ that adds up to t iff
  there is a $P' \subset S'$ that adds up to $(\Sigma S')/2 = 2n$.

The reduction

- Let ρ map (S, t) to $S' =_{\text{df}} S \cup \{n+t,\, 2n-t\}$.
  So S' adds up to $4n$.
- Claim: There is a $P \subseteq S$ that adds up to t iff
  there is a $P' \subset S'$ that adds up to $(\Sigma S')/2 = 2n$.
- $\Rightarrow$: If there is a $P \subset S$ that adds up to t then
  then $P \cup \{2n-t\}$ is a subset of S'
  that adds up to $t + (2n-t) = 2n = (\Sigma S')/2$.

The reduction

- Let ρ map (S,t) to $S' =_{\mathrm{df}} S \cup \{n+t,\, 2n-t\}$.

  So S' adds up to $4n$.

- Claim: There is a $P \subseteq S$ that adds up to t iff

  there is a $P' \subset S'$ that adds up to $(\Sigma S')/2 = 2n$.

- $\Rightarrow$: If there is a $P \subset S$ that adds up to t then

  then $P \cup \{2n-t\}$ is a subset of S'

  that adds up to $t + (2n-t) = 2n = (\Sigma S')/2$.

- $\Leftarrow$: If there is a $P' \subset S'$ that adds up to $\Sigma S' = 2n$

  then $S' - P'$ adds up to $4n - 2n = 2n$.

  The new elements add up to $3n$,

  so each of P' and $S' - P'$ has one of the two.

The reduction

- Let ρ map (S,t) to $S' =_{\text{df}} S \cup \{n+t,\, 2n-t\}$.
  So S' adds up to $4n$.
- Claim: There is a $P \subseteq S$ that adds up to t iff
  there is a $P' \subset S'$ that adds up to $(\Sigma S')/2 = 2n$.
- $\Rightarrow$: If there is a $P \subset S$ that adds up to t then
  then $P \cup \{2n{-}t\}$ is a subset of S'
  that adds up to $t + (2n-t) = 2n = (\Sigma S')/2$.
- $\Leftarrow$: If there is a $P' \subset S'$ that adds up to $\Sigma S' = 2n$
  then $S'{-}P'$ adds up to $4n{-}2n = 2n$.
  The new elements add up to $3n$,
  so each of P' and $S'{-}P'$ has one of the two.
- Removing $2n-t$ from the half that has it,
  yields a $P \subset S$ that adds up to t.

Lessons

- Reductions may be ingenious,

  using particulars of the problems compared.

  There are no silver bullets.

Lessons

- Reductions may be ingenious,

  using particulars of the problems compared.

  There are no silver bullets.

- ***Warning:*** We only had S and t given.

  S' was calculated, but P & P' were ***hypothetical***,

  linking the property of the source-problem to the

  property of the target-problem.

NP COMPLETENESS

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:

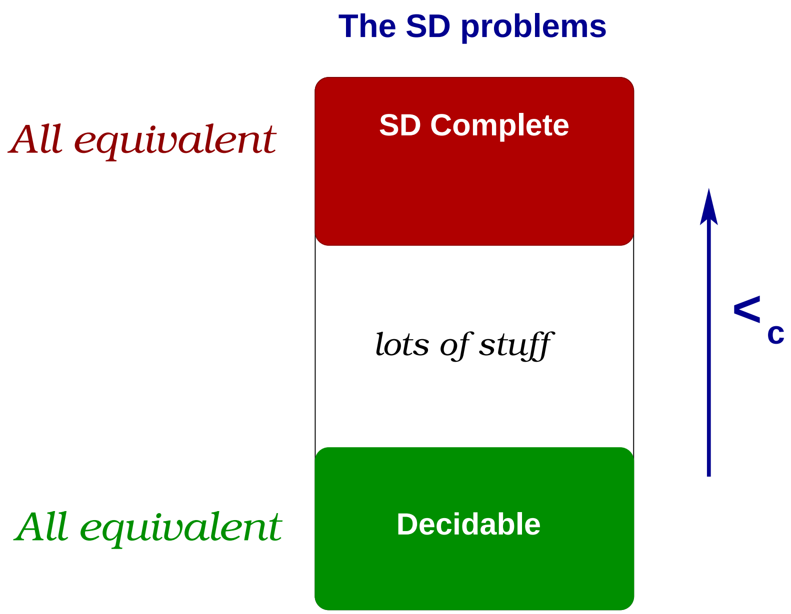
  Every SD problem $\mathcal{Q}$ is reducible to $\mathcal{P}$ since $\mathcal{P}$ is SD-hard.

  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: **ACCEPT**.

Clear broad picture for SD...



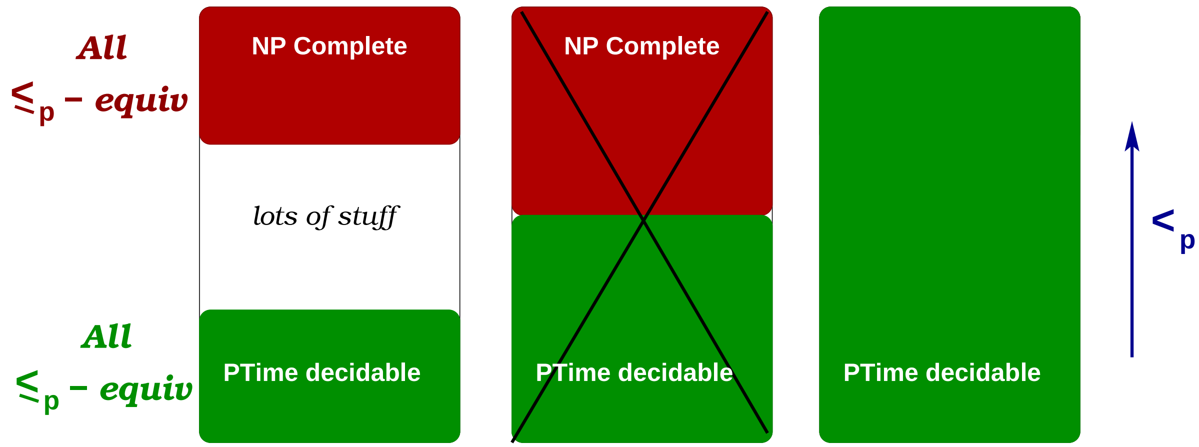
- Whether there is something in the middle was an open problem for about two decades, until proven by Albert Nuchnik (1956) and independently by Richard Friedman (1957).
- Subsequent research showed that there is quite a lot there...

The analog: 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



Impossible
(Ladner's Theorem)

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 **HAMILT-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 E 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 E evaluates to $V(E)=0$ or $V(E)=1$.

- Example: If $V(x)=0, V(y)=0$ then $V(-x \wedge -(y \vee x))=1$,

  but if $V(x)=1$ then $V(-x \wedge -(y \vee x))=0$.

- A valuation V ***satisfies*** E if $V(E)=1$.

- E is ***satisfiable*** if it is satisfied by ***some*** V ,

  It is ***valid*** if it satisfied by ***every*** V.

- So E is satisfiable iff $-E$ is not satisfiable

  and is valid iff $-E$ is not satisfiable.

Boolean satisfiability

- **BOOL-SAT**: Given a boolean expression, is it satisfiable?
- A certification for **BOOL-SAT**:

  the certificate for E is a valuation satisfying it.
- Checking a certificate is PTime in the size of the expression.

  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.

Playing Charades with decision-problems

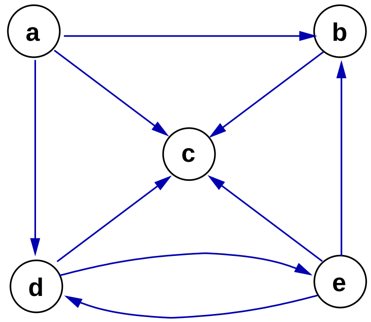
Boolean coding of Hamiltonian-Path

- HAMILTONIAN-PATH: *Given directed graph $G = (V, E)$, does it have a path visiting every vertex once.*
- I.e. is there a listing $u_1, u_2, \ldots, u_n$ of the vertices s.t. $u_i(E)u_{i+1}$ for $i < n$.
- Convey this by a boolean expression.

  For each $v \in V$ and $i = 1..n$ a fresh boolean variable x_{iv} intended to be true iff u_i is v.

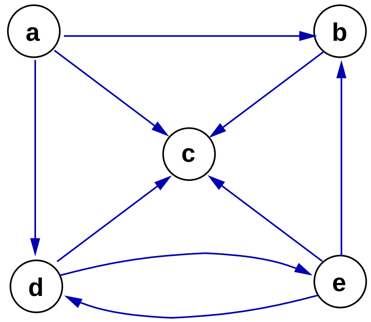
The boolean expression

- Example:



The boolean expression

- Example:

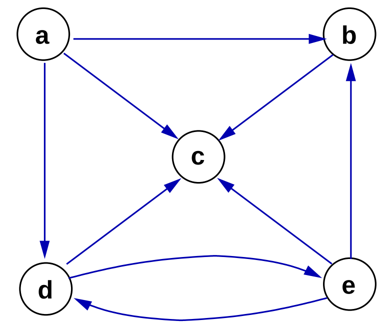


- A listing a, b, c, d, e is conveyed by the valuation assigning 1 to x_{1a}, x_{2b}, x_{3c}, x_{4d}, x_{5e} and 0 to the other variables:

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

The boolean expression

- Example:



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

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

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 $-(x_{iv} \wedge x_{jv})$

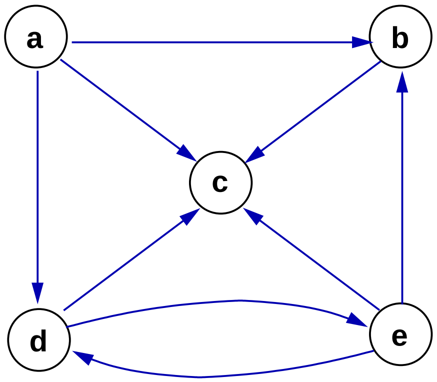
Successive vertices are adjacent in the graph

- 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 $\rho : \ \mathbf{HP} \ \leqslant_p \ \mathbf{BOOL\text{-}SAT}$
- ρ maps a directed graph $G = (V, E)$ 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 $u_1 \rightarrow \cdots \rightarrow u_n$ 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**

SAT IS NP-COMPLETE

Boolean coding of PTime

- Let M be a Turing acceptor over Σ
  running within time $f(n)$ (f a polynomial).
  We'll assume $f(n) \geqslant n$.
- Define a reduction $\rho : \mathcal{L}(M) \leqslant_p \mathbf{BOOL\text{-}SAT}$
- ρ maps each Σ-string w
  to a boolean-expression E_w such that

  M *accepts* w IFF E_w *is satisfiable.*

The trace displayed as a square grid

M is $\quad \mathtt{s} \xrightarrow{>\,(+)} \mathtt{r} \quad \mathtt{r} \xrightarrow{1\,(+)} \mathtt{p} \quad \mathtt{q} \xrightarrow{0\,(+)} \mathtt{q} \quad \mathtt{q} \xrightarrow{\sqcup\,(-)} \mathtt{a}$

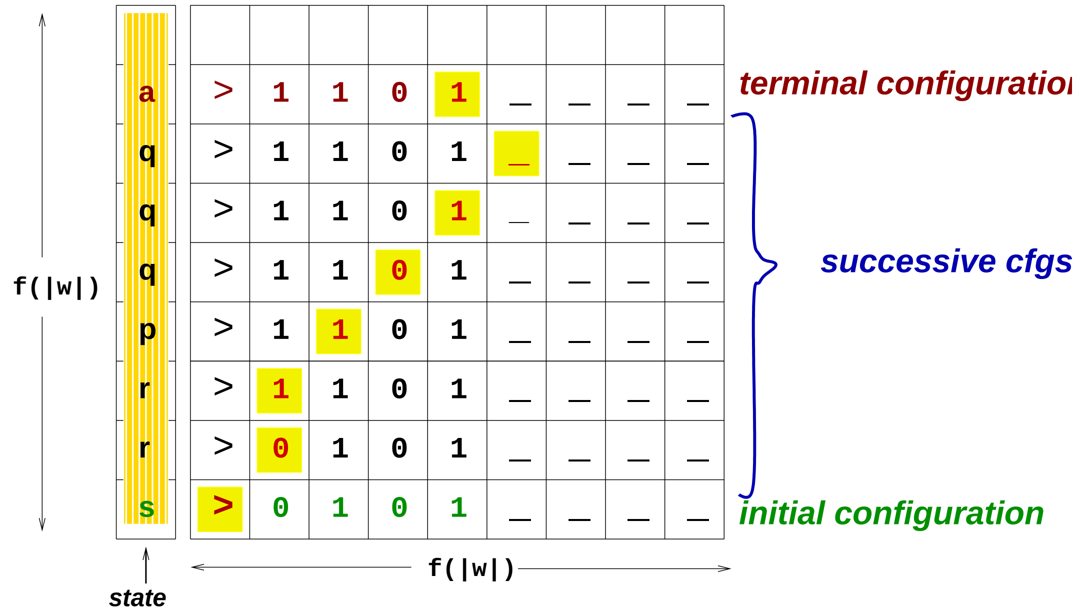
$\quad \mathtt{r} \xrightarrow{0\,(1)} \mathtt{r} \quad \mathtt{p} \xrightarrow{1\,(+)} \mathtt{q} \quad \mathtt{q} \xrightarrow{1\,(+)} \mathtt{q}$

The trace displayed as a square grid

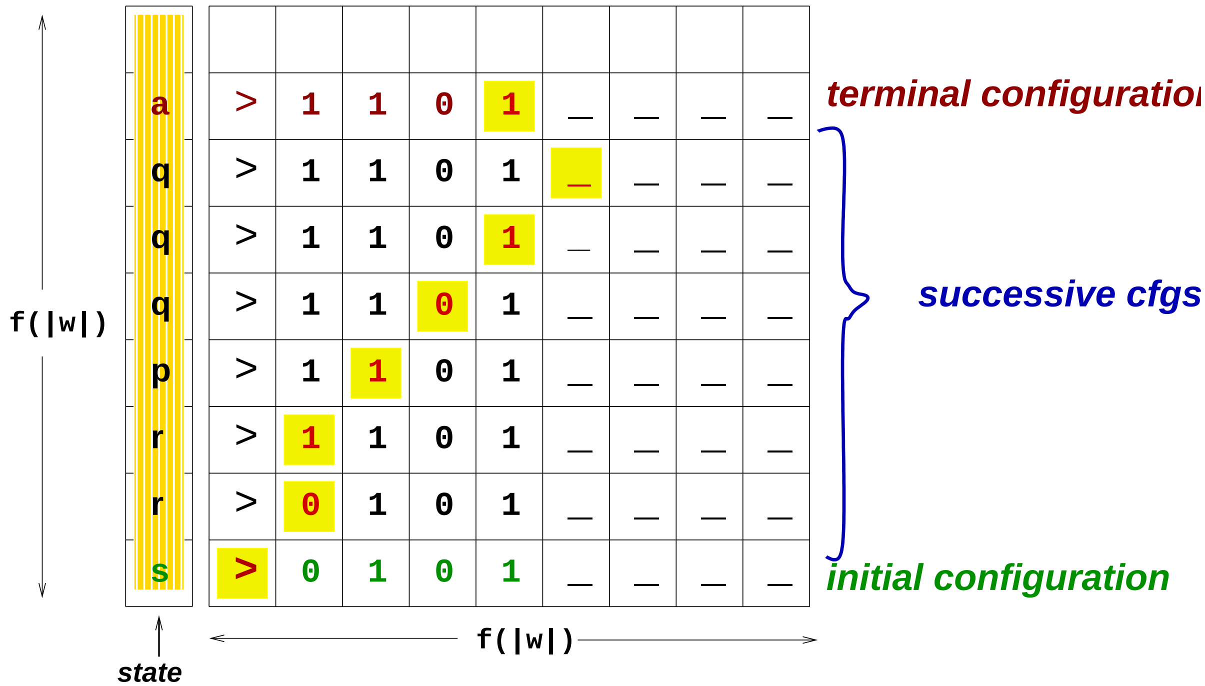
M is $\quad \mathtt{s} \xrightarrow{>\,(+)} \mathtt{r} \quad \mathtt{r} \xrightarrow{1\,(+)} \mathtt{p} \quad \mathtt{q} \xrightarrow{0\,(+)} \mathtt{q} \quad \mathtt{q} \xrightarrow{\sqcup\,(-)} \mathtt{a}$

$\quad \mathtt{r} \xrightarrow{0\,(1)} \mathtt{r} \quad \mathtt{p} \xrightarrow{1\,(+)} \mathtt{q} \quad \mathtt{q} \xrightarrow{1\,(+)} \mathtt{q}$

Trace of M for input w:



The grid defined by yes/on questions



- We'll have a collection of boolean variables,

  each standing for a question about the trace of M for input w.

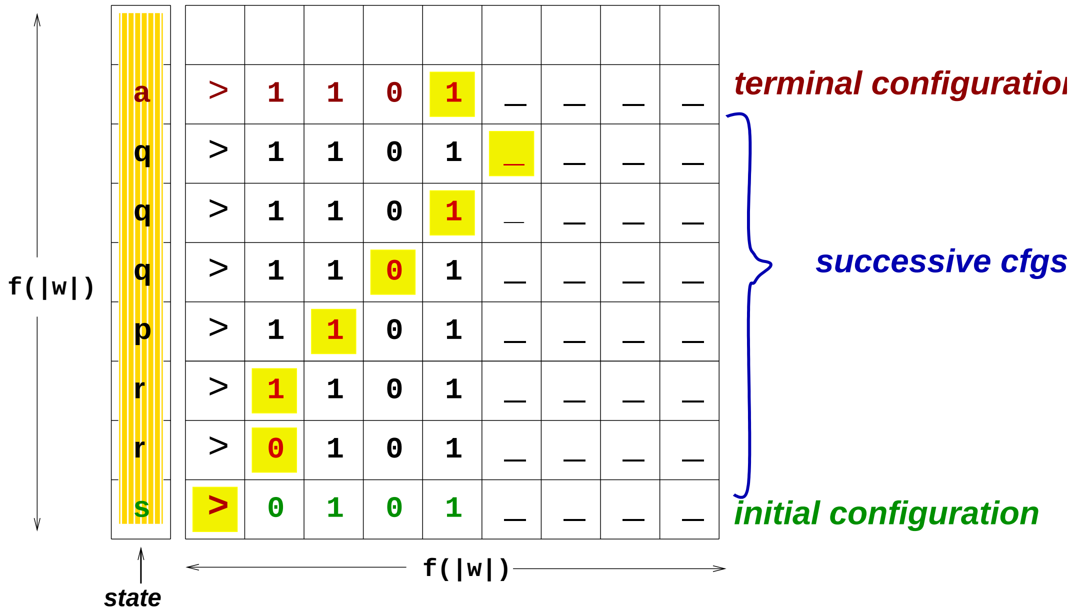
  (A session of the party game *charades*.)

- For each state q and row $i \leqslant |w|$ $x_{i,q}$ for *"state of i'th cfg is q"*

  Examples: $x_{1,s}$, $x_{4,p}$

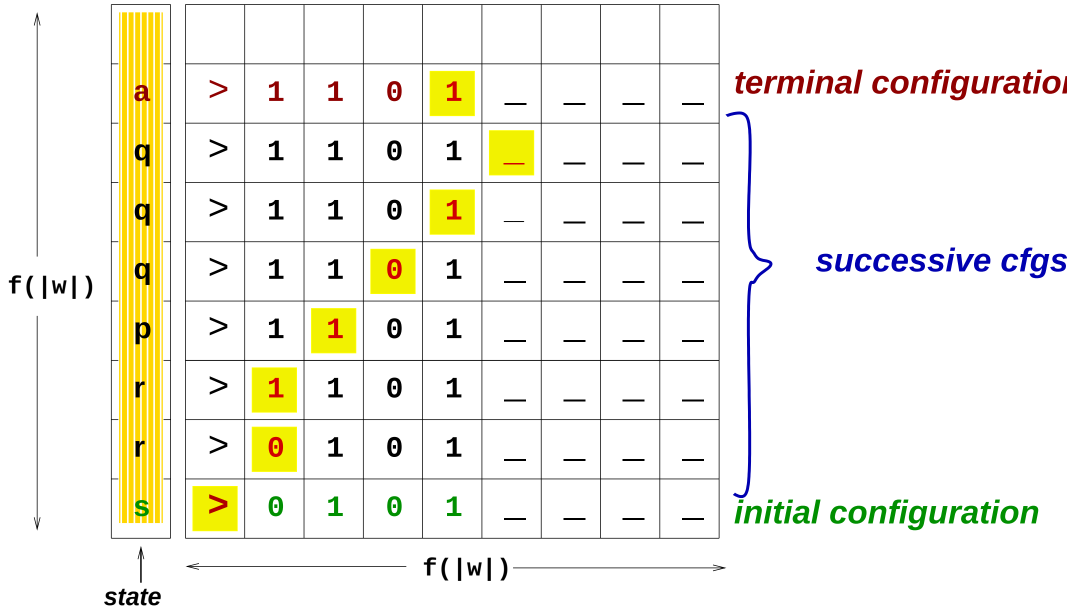
- For each $i, j \leqslant |w|$: $c_{i,j}$ for *"cursor of i'th cfg at j"*
- For each $i, j \leqslant |w|$ and $\sigma \in \Sigma$: $\ell_{i,j,\sigma}$ for *"(i, j) cell has σ*

Yes/no for consistency conditions



- One state + one cursor per row
- one symbol per cell
- First row is initial state + $> w \sqcup^m$.
- 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
- Analogous to the edge-condition for HAMILTONIAN-PATH.
- The initial cfg has state s and cursored string in $\underline{>}w\sqcup^*$.
- Successive cfgs are related by the transitions of M (or repeat terminal cfgs).

- Example: For $q \xrightarrow{\sigma(\tau)} r$

$$
\begin{aligned}
&(x_{i,q} \wedge c_{i,j} \wedge \ell_{i,j,\sigma} \\
&\quad \to \\
&\quad p_{i+1,r} \ \wedge\ c_{i+1,j} \ \wedge\ \ell_{i+1,j,\tau} \\
&\quad \wedge \quad \textstyle\bigwedge_{k \neq j} \ \ell_{i,k,\xi} \ \to\ \ell_{i+1,k,\xi}
\end{aligned}
$$

- The ***accept*** state appears: $\bigvee_i x_{i,\mathbf{a}}$.

Coding PTime certification

- Consider a PTime-certified language L,

  with a feasible certification $\vdash$.

- That is, $c \vdash w$ is decided by a Turing-acceptor M,

  in time $\leqslant f(|w|)$, with $|c| \leqslant f|w|)$ (f polynomial).

- That is, $w \in L$ iff M accepts w, c for some c of length $\leqslant f(|w|)$ in time $\leqslant f(|w|)$. (The comma is a separator-symbol).

- We cannot construct a trace-layout for w,

  because we don't have c:

  The values of the boolean variables $\ell_{1j\sigma}$

  are unknown for $j > |w|$.

Boolean satisfiability is NP-complete

- But the satisfiability of the resulting boolean expression E_w
  means precisely that w, c is accepted by M for ***some*** such values!
- The satisfiability of E_w is equivalent to
  M accepting w, c for ***some*** c of size $\leqslant g(|w|)$
  in time $\leqslant f(x)$.
- We this proved that BOOL-SAT *is NP-hard.*
- Since BOOL-SAT is PTime certified we conclude:
  Theorem. BOOL-SAT *is NP-complete.*

NP-COMPLETENESS OF ADDITIONAL PROBLEMS

Normal forms

- We can now prove that certain problems $\mathcal{P}$

  are NP-complete and therefore dangerously complex,

  by defining a PTime reduction $\textbf{BOOL-SAT} \leqslant_p \mathcal{P}$.

- Defining such reductions may be challenging,

  because boolean expressions can be arbitrarily complex.

  Can we facilitate reductions by focusing on some that are

- Reductions to $\boxed{\textit{normal forms}}$ are all around!

Normal forms

- We can now prove that certain problems $\mathcal{P}$
  are NP-complete and therefore dangerously complex,
  by defining a PTime reduction $\textbf{BOOL-SAT} \leqslant_p \mathcal{P}$.
- Defining such reductions may be challenging,
  because boolean expressions can be arbitrarily complex.
  Can we facilitate reductions by focusing on some that are
- Reductions to ***normal forms*** are all around!
- Decimal fractions (percents): $3/8$ versus $4/11$ (.375 vs .364)

Normal forms

- We can now prove that certain problems $\mathcal{P}$

  are NP-complete and therefore dangerously complex,

  by defining a PTime reduction $\textbf{BOOL-SAT} \leqslant_p \mathcal{P}$.

- Defining such reductions may be challenging,

  because boolean expressions can be arbitrarily complex.

  Can we facilitate reductions by focusing on some that are

- Reductions to $\boxed{\textbf{\textit{normal forms}}}$ are all around!

- Decimal fractions (percents): $3/8$ versus $4/11$ (.375 vs .364)

- Better: Normalized scientific notation for real numbers:

  $123.45 = 1.2345 \times 10^2$, $0.0012345 = 1.2345 \times 10^{-3}$

- Displays immediately the order of magnitude.

Normal forms

- We can now prove that certain problems $\mathcal{P}$

  are NP-complete and therefore dangerously complex,

  by defining a PTime reduction $\textbf{BOOL-SAT} \leqslant_p \mathcal{P}$.

- Defining such reductions may be challenging,

  because boolean expressions can be arbitrarily complex.

  Can we facilitate reductions by focusing on some that are

- 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:

  $123.45 = 1.2345 \times 10^2$, $0.0012345 = 1.2345 \times 10^{-3}$

- Displays 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 **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:

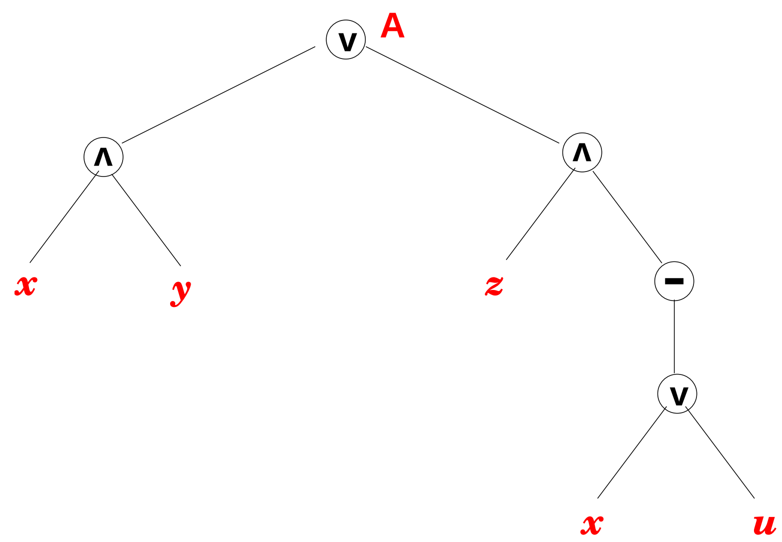
  CNF-SAT $\leqslant_p \mathcal{P}$ easier to show than **SAT** $\leqslant_p \mathcal{P}$.

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 $\boxed{\textit{3CNF}}$ expressions where each clause has $\leqslant 3$ literals.

3CNF-Satisfiability

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



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

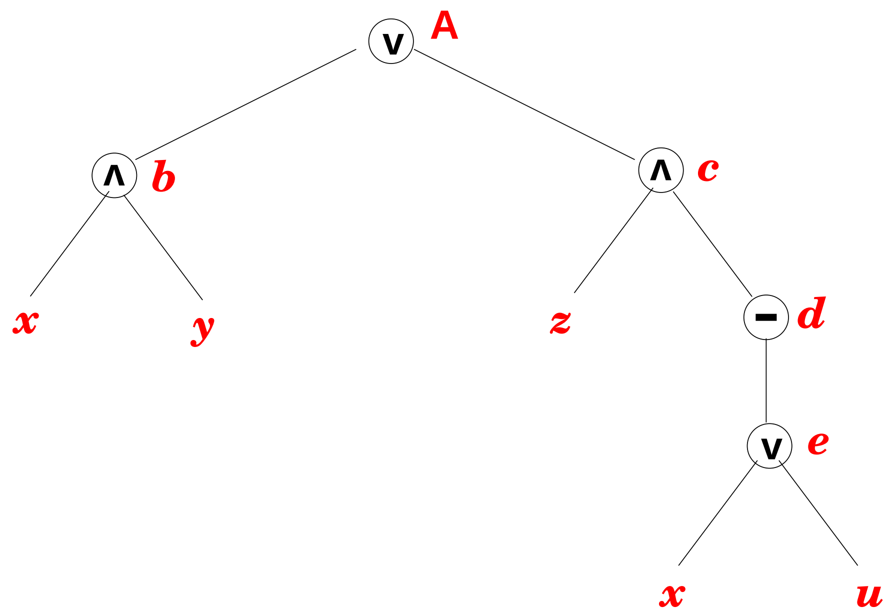
$$a \equiv A$$

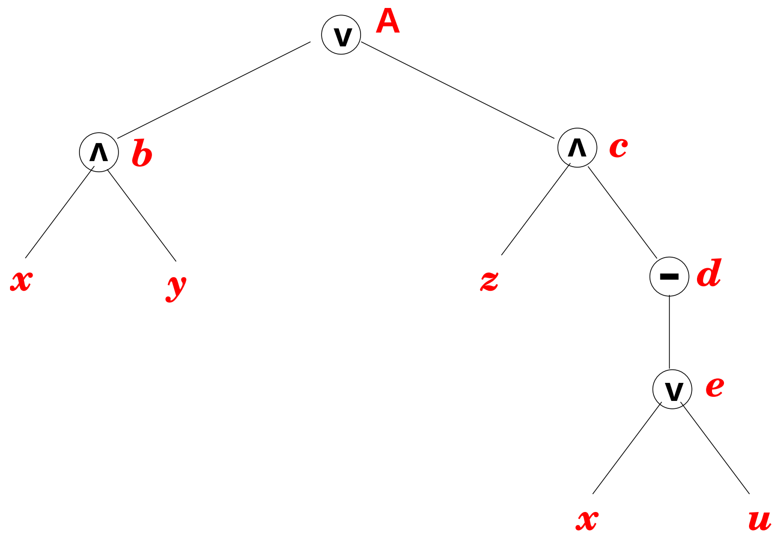
$$b \equiv x \wedge y$$

$$c \equiv x \wedge u$$

$$d \equiv -(x \wedge u)$$

$$e \equiv z \vee -(x \wedge u)$$





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

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

Equivalence	in 3CNF format
$(a \leftrightarrow (b \vee c))$	$\bar{a} \vee b \vee c$
	$a \vee \bar{b}$
	$a \vee \bar{c}$
$(b \leftrightarrow (x \wedge y))$	$\bar{b} \vee x$
	$\bar{b} \vee y$
	$\bar{x} \vee \bar{y} \vee b$
$(c \leftrightarrow (z \wedge d))$	$\bar{c} \vee z$
	$\bar{c} \vee d$
	$\bar{z} \vee \bar{d} \vee c$
$(d \leftrightarrow -e)$	$\bar{d} \vee \bar{e}$
	$e \vee \bar{d}$
$(e \leftrightarrow (x \vee u))$	$\bar{e} \vee x \vee u$
	$\bar{x} \vee e$
	$\bar{u} \vee e$

- 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

     $(L_0 \vee L_1 \vee y) \ \wedge \ (L_0 \vee L_1 \vee \bar{y}) \quad$ (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 $\rho :$ 3CNF-SAT $\leqslant_p$ INDEP-SET.

  Map a 3CNF expression E with k clauses

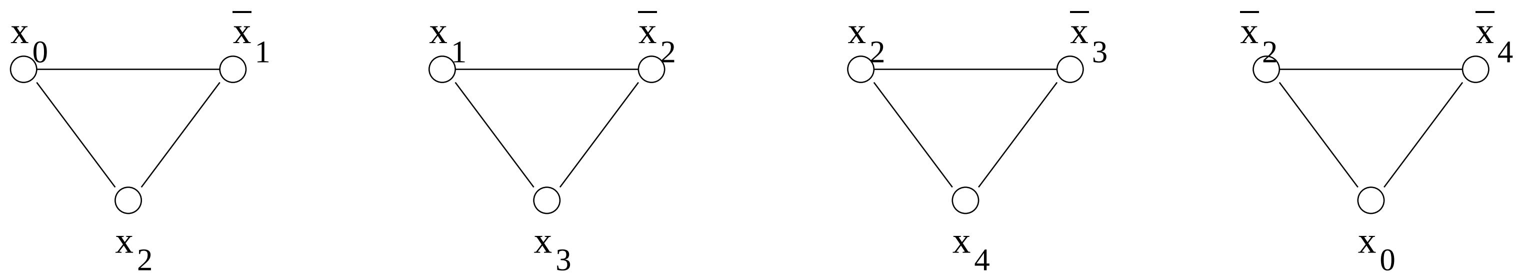
  to graph G + target k.

- A thought: each clause is mapped 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 :

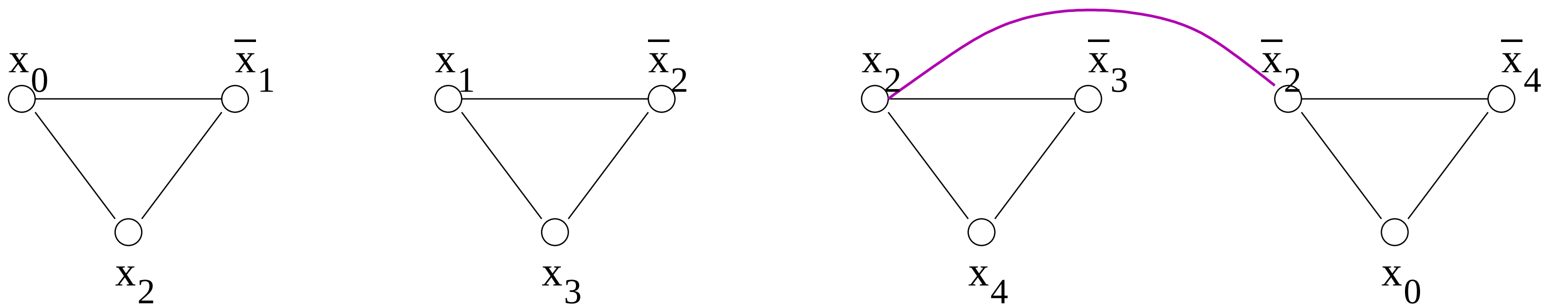


- Choose a vertex in each triangle, eg top left.

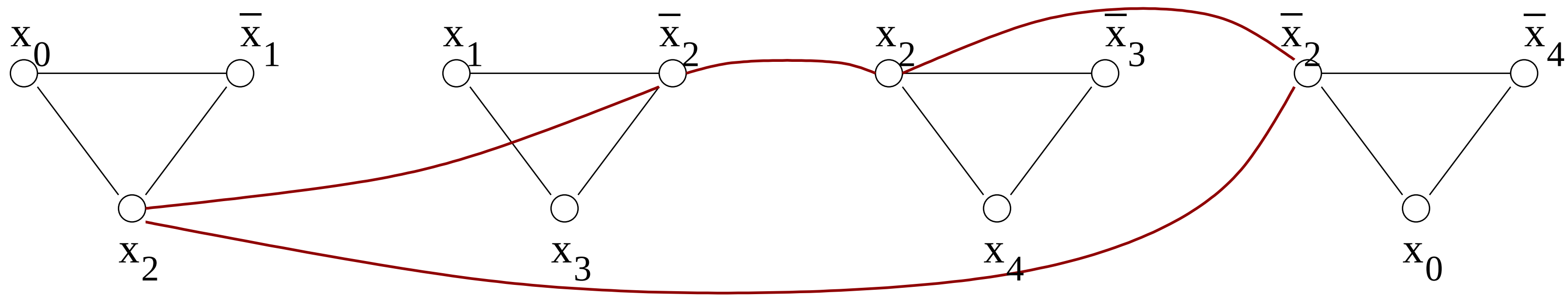
  Oops, we are trying to have both x_2 and $\bar{x}_2$ true!

Add consistency edges

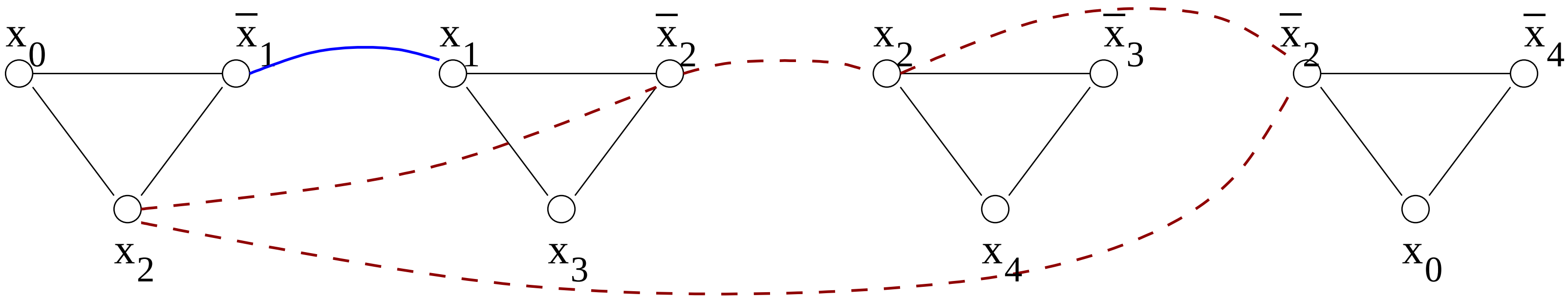
- Consistency edge for x_2:



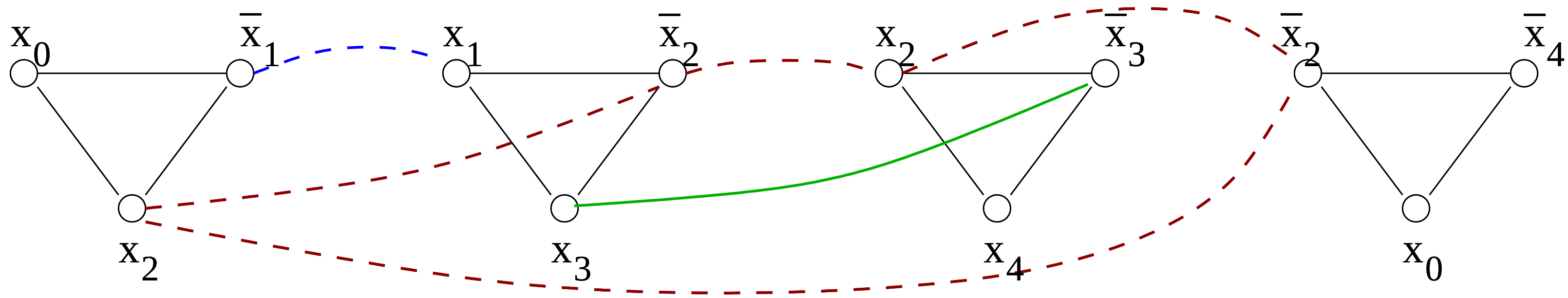
- Additional consistency edges for x_2:



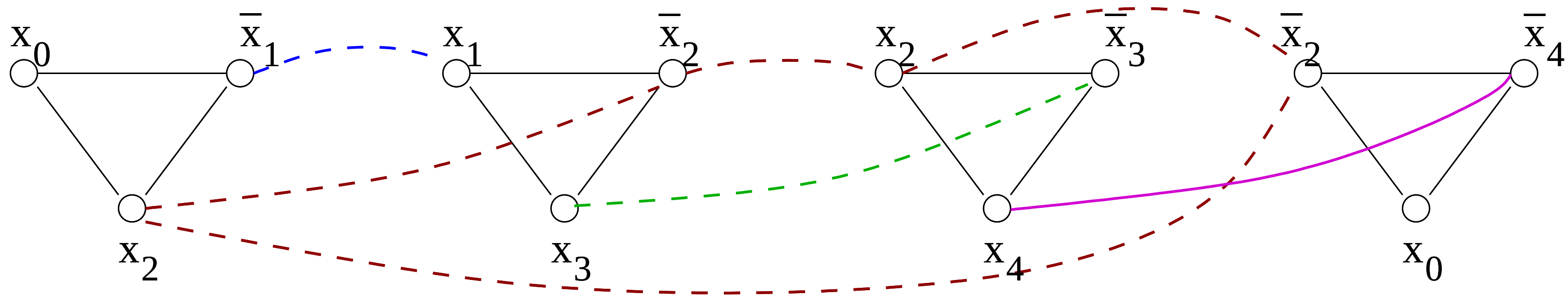
- Consistency edge for x_1:



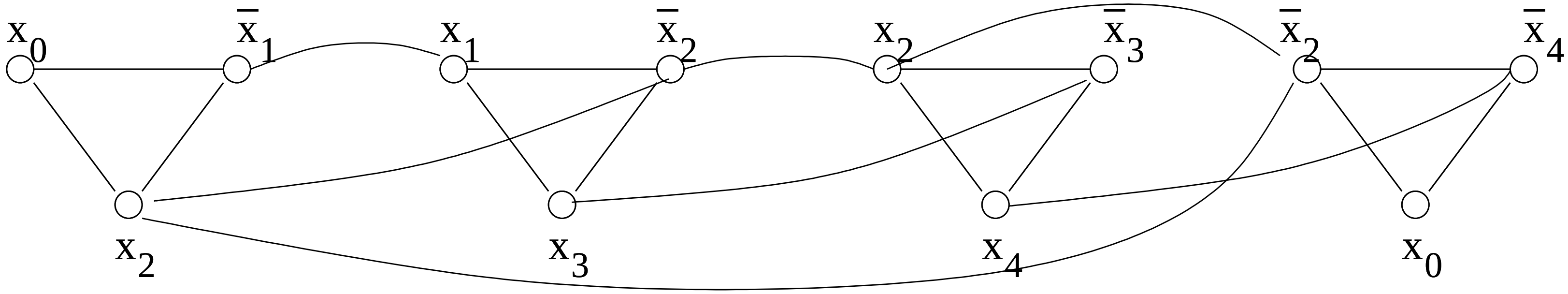
- Consistency edge for x_3:



- 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 satisfies the labels of S satisfies A.

Consequence: **clique** *is NP-complete*

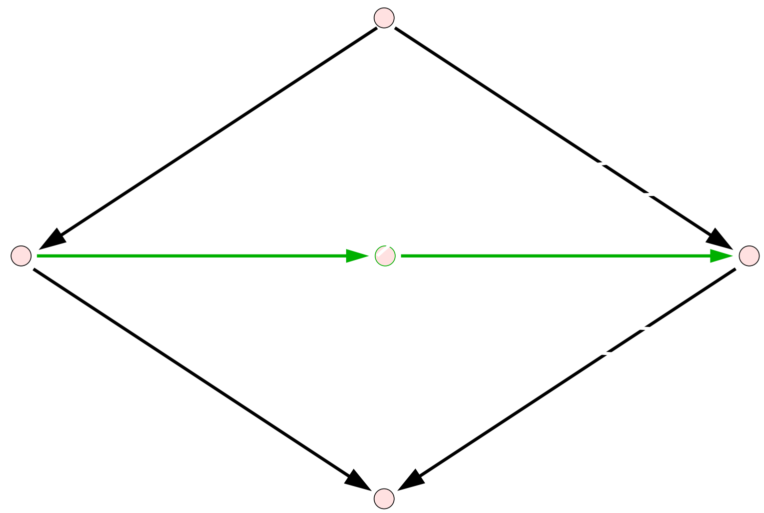
- We showed that **CLIQUE** is NP.
- **INDEP-SET** $\leqslant_p$ **CLIQUE**
- Since **INDEP-SET** is NP-hard, so is **CLIQUE**.

EAMPLE: DIRECTED HAMITONIAN PATH

Hamiltonian-path *is NP-complete*

- HAMILTONIAN-PATH (H-PATH): *Given a directed graph* $G = (V, E)$,
  does it have a path visiting each vertex exactly once.
- H-PATH has a feasible certification:
  the certificate is the path.
- To prove NP-hardness show 3CNF-SAT $\leqslant_p$ H-PATH

A diamond gadget

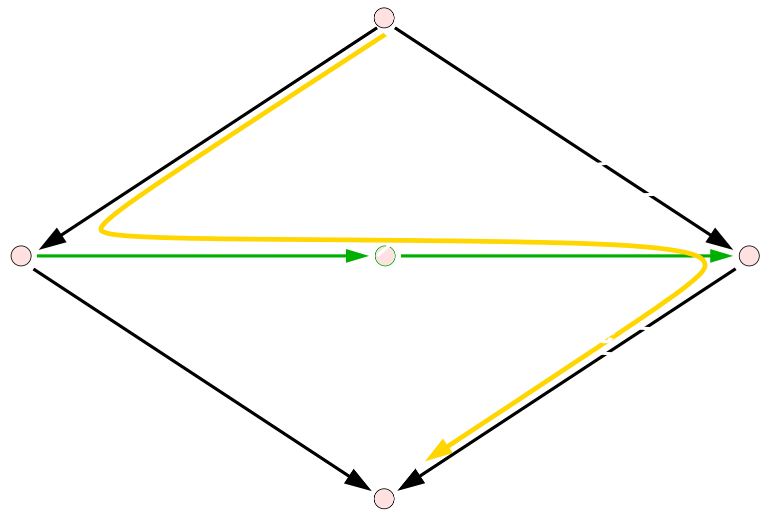


- This is an example of a ***gadget***:

  a component, often repeated, of a compound discrete object.

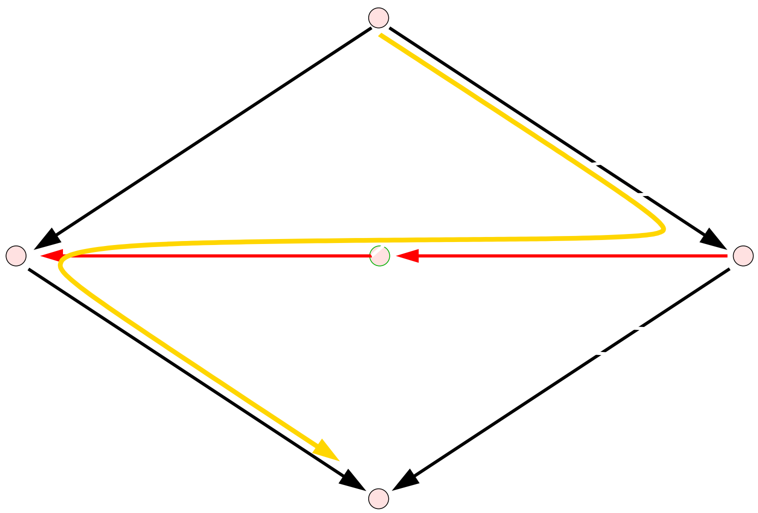
- Does it have an H-path?

A diamond gadget



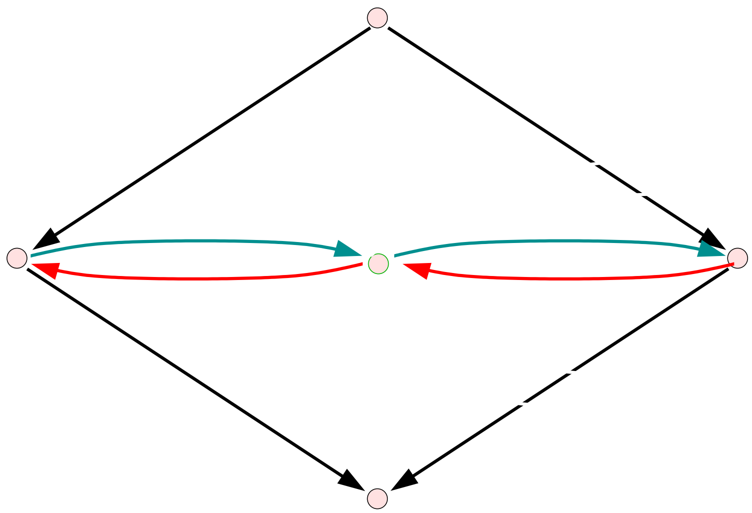
- An H-path through the gadget must follow the rightward edges.

A diamond gadget



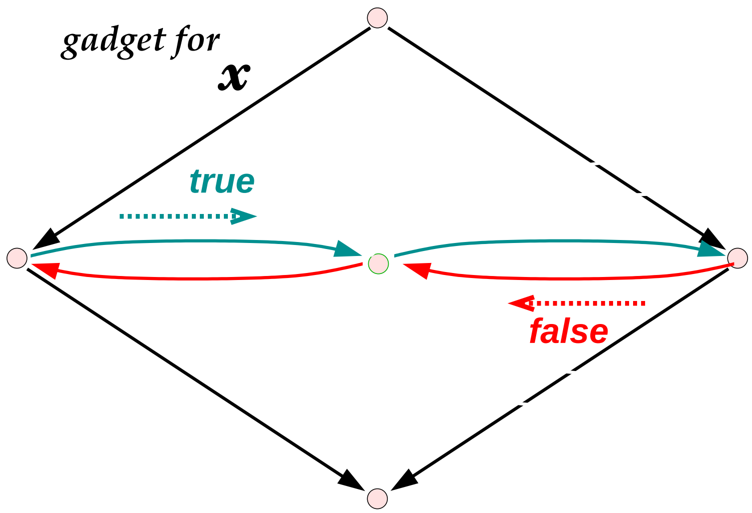
- Dually for leftwards horizontal edges.

A diamond gadget



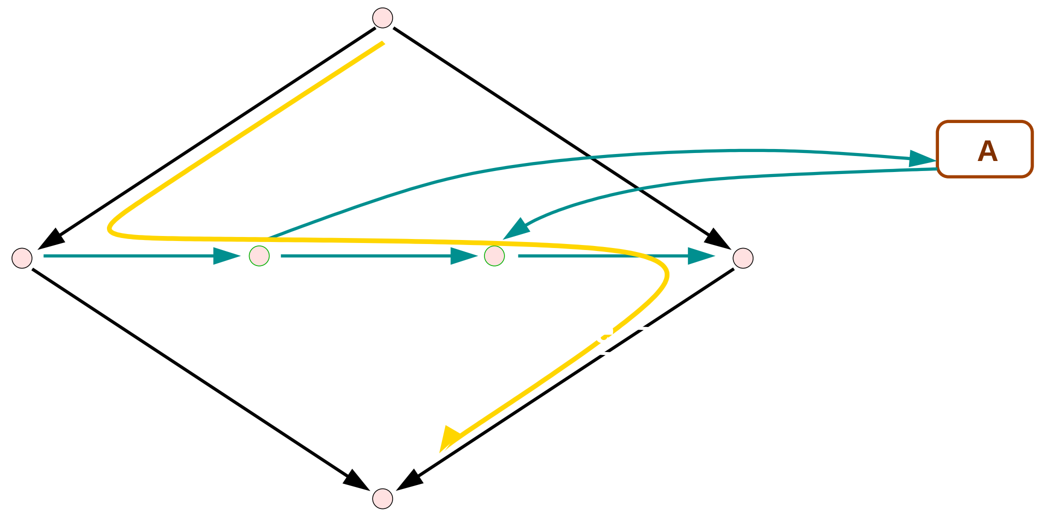
- With edges pointing both ways we get a choice between two H-paths.

The gadget as a boolean switch



- Take each choice of H-path to represent a truth value of a boolean variable x.

Side trips of a H-path



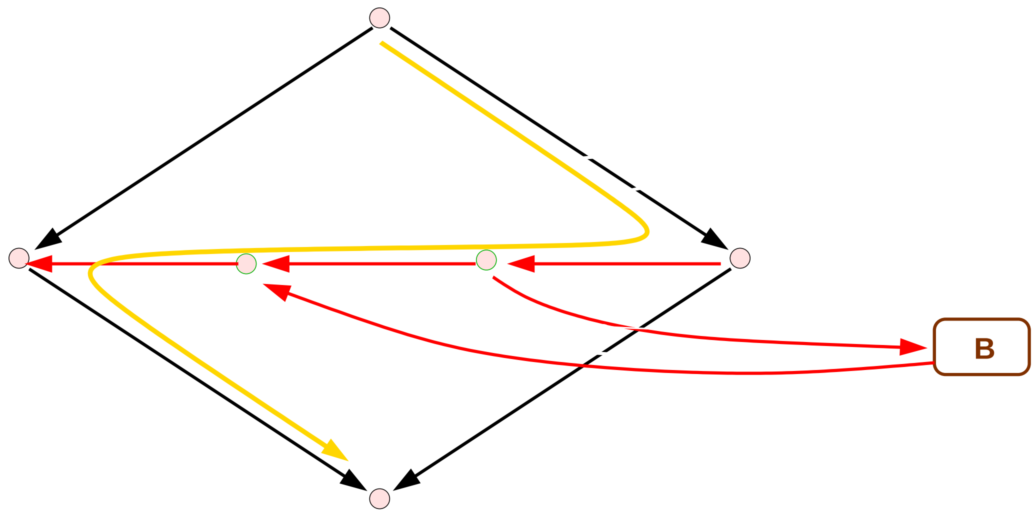
- Ahamilt-bool-basic-use-positiven H-path crossing the gadget rightwards (x ***true***)

  can optionally veer to visit an extenal vertex A

  and return one step to the right.

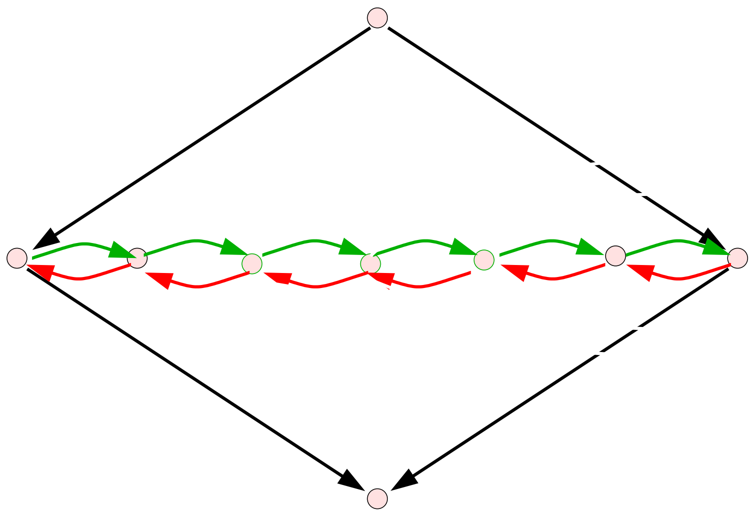
- Not so for an H-path for ***false***.

Side trips of a H-path



- Dually, an H-path crossing the gadget leftwards (x ***false***) can veer to visit edge B and return one step to the left.

Side trips of a H-path



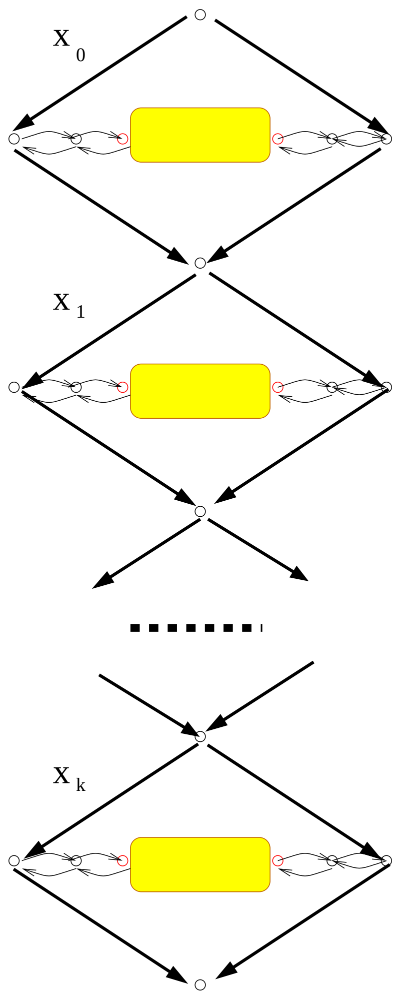
- To visit up to n external vertices

  the horizontal "switch-box" must have

  at least $n+1$ vertices (endpoints included).

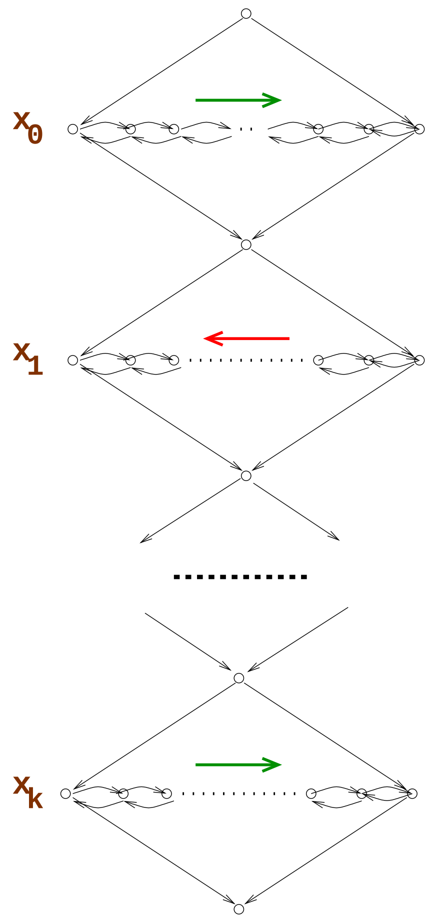
A serial panel of gadgets

For variables $x_1, \ldots x_k$ we form a serial panel of k gadgets.



A serial panel of gadgets

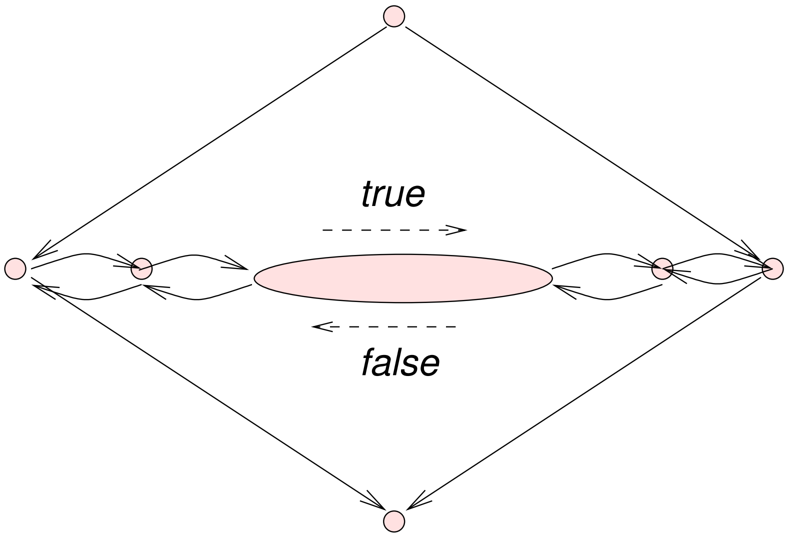
2^k possible H-paths, each representing a unique boolean valuation:



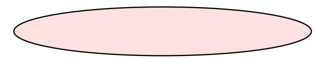
represents the valuation

x_0	x_1	...	x_k
$\downarrow$	$\downarrow$		$\downarrow$
1	0		1

Here are the constraints



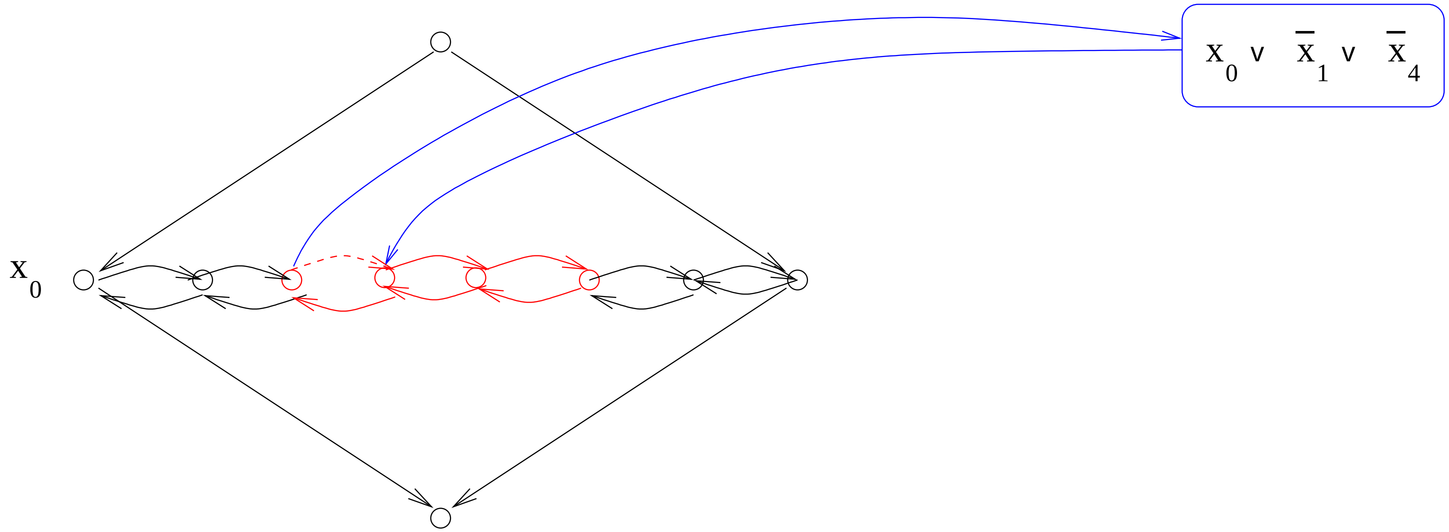
the sub−graph



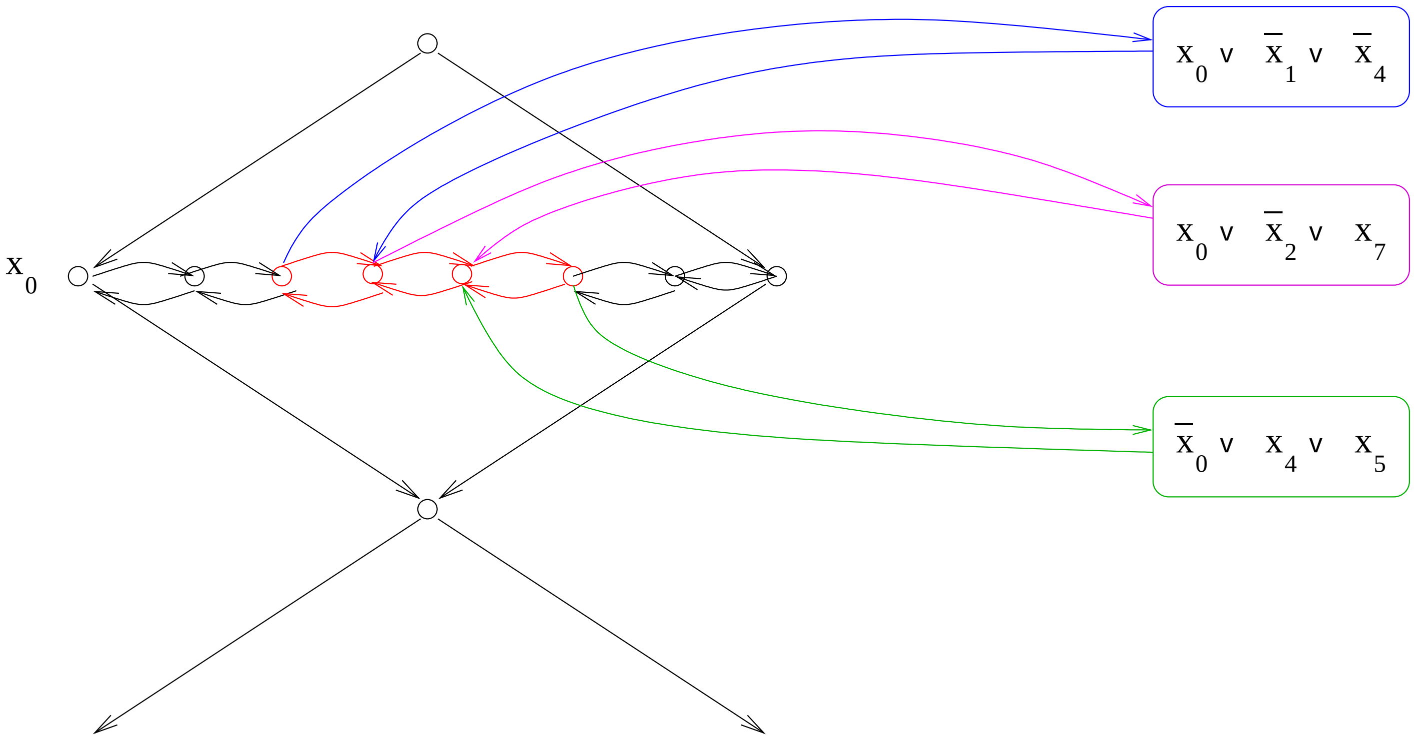
is a "switchboard" for boolean−dependent side trips

- Given a 3CNF expression E with variables $x_1 \ldots x_k$sider sequentla panel of
- Each clause represented by a vertex.
- Satifying a clause represented by visiting it.
- Every clause risited by one literal (Hamiltonian!)

Variable x_0 visits a clause

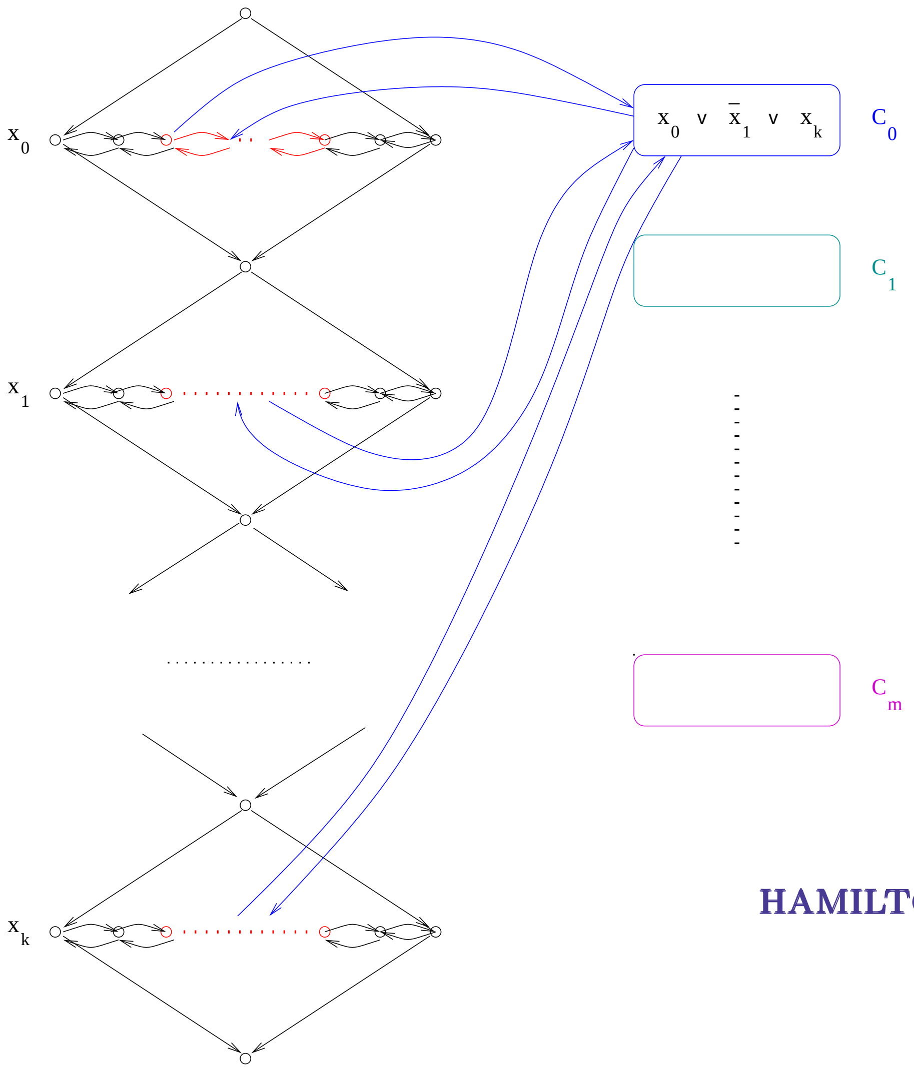


* *Variable* $x+0$ *visits multiple clauses*



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

Combining the switchboards



HAMILTONIAN-PATH is NP-complete.