(u,v).f \quad f(u,v)

v.d \quad d(v)

NP-complete

Class of problems for which either all have poly-time algns. or none have poly algns.
Program to test if a binary number is even. Input is $\$\$ termina
Output is written immediately after $\$, 1 for yes, 0 for no.

- Read until $\$ (state $q_0$)
- Back up, check last digit (state $q_1$)
- if even, write a 1 (states $q_2$, $q_3$, $q_F$)
- if odd, write a 0 (states $q_4$, $q_5$, $q_F$)

Here is a program. Each cell is (new state, write symbol move)

| state | input 0 | input 1 | input $\$
|-------|---------|---------|---------|
| $(q_0)$ | $(q_0, -, R)$ | $(q_0, -, R)$ | $(q_1, -, L)$
| $(q_1)$ | $(q_2, -, R)$ | $(q_1, -, R)$ | error
| $(q_2)$ | error | error | $(q_3, -, R)$
| $(q_3)$ | $(q_F, 1, -)$ | $(q_F, 1, -)$ | $(q_F, 1, -)$
| $(q_4)$ | error | error | $(q_5, -, R)$
| $(q_5)$ | $(q_F, 0, -)$ | $(q_0, 0, -)$ | $(q_0, 0, -)$
| $(c_F)$ | halt | halt | halt
Any computer/compiler can be "simulated" on a turing machine with only a poly amt. of slowdown.

P is invariant to compiler issues

Is L sorted? 2, 7, 10, 12

\[
\begin{align*}
S &= \{s, g, t\} \\
S &= \{s, b, t\} \quad \text{NO} \\
S &= \{s, b, a, t\} \quad \text{NO}
\end{align*}
\]
\[
\phi = (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \land (x_2 \lor x_4) \land (x_1 \lor \overline{x_3})
\]

**IS there a way to set vars. so that \(\phi\) is true?**

**YES**

\[
(x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2) \land (\overline{x_1} \lor \overline{x_2})
\]

**NO**

---

**Reductions**

\(Y \leq X\) \(\text{we can use } X \text{ as a "substitute for solving } Y\) 

\(Y\) is subtraction of two ints \(\text{(negation)}\)

\(X\) is addition of two ints \(\text{(easier)}\)

\((6,4)\) you know how to do your friend knows how to add.

want to compute \(6 - y\)

- Alg. Negate the \(4 \rightarrow (6, -4)\) give it to your friend.
- Friend return \(6 + (-4) \rightarrow 2\).
- You have 2.
**Definition** \( Y \leq X \) means

- \( Y \) is polynomial time reducible to \( X \), which means
  
  there exists a polynomial time computable function \( f \) that maps inputs to \( Y \) to inputs to \( X \), such that

  input \( y \) to problem \( Y \) returns \textbf{"Yes"} iff input \( f(y) \) to problem \( X \) returns \textbf{"Yes"}

**Informally** \( Y \leq X \) means that \( Y \) is \textbf{“not much harder than”} (\textbf{“easier than”}) \( X \)

\[
y \mapsto f(y) \quad \text{input to } Y \quad \mapsto \frac{y \in S}{y \not\in S}
\]

**Theorem**

If \( Y \leq X \) then \( X \in P \Rightarrow Y \in P \)

**Contrapositive**

If \( Y \leq X \) then \( Y \not\in P \Rightarrow X \not\in P \)
Theorem  SAT is NP-complete

Proof idea: The turing machine program for any problem in NP can be verified by a polynomial sized SAT instance that encodes that the input is well formed and that each step follows legally from the next.

Implication We now have one NP-complete problem. We will now reduce other problems to it.