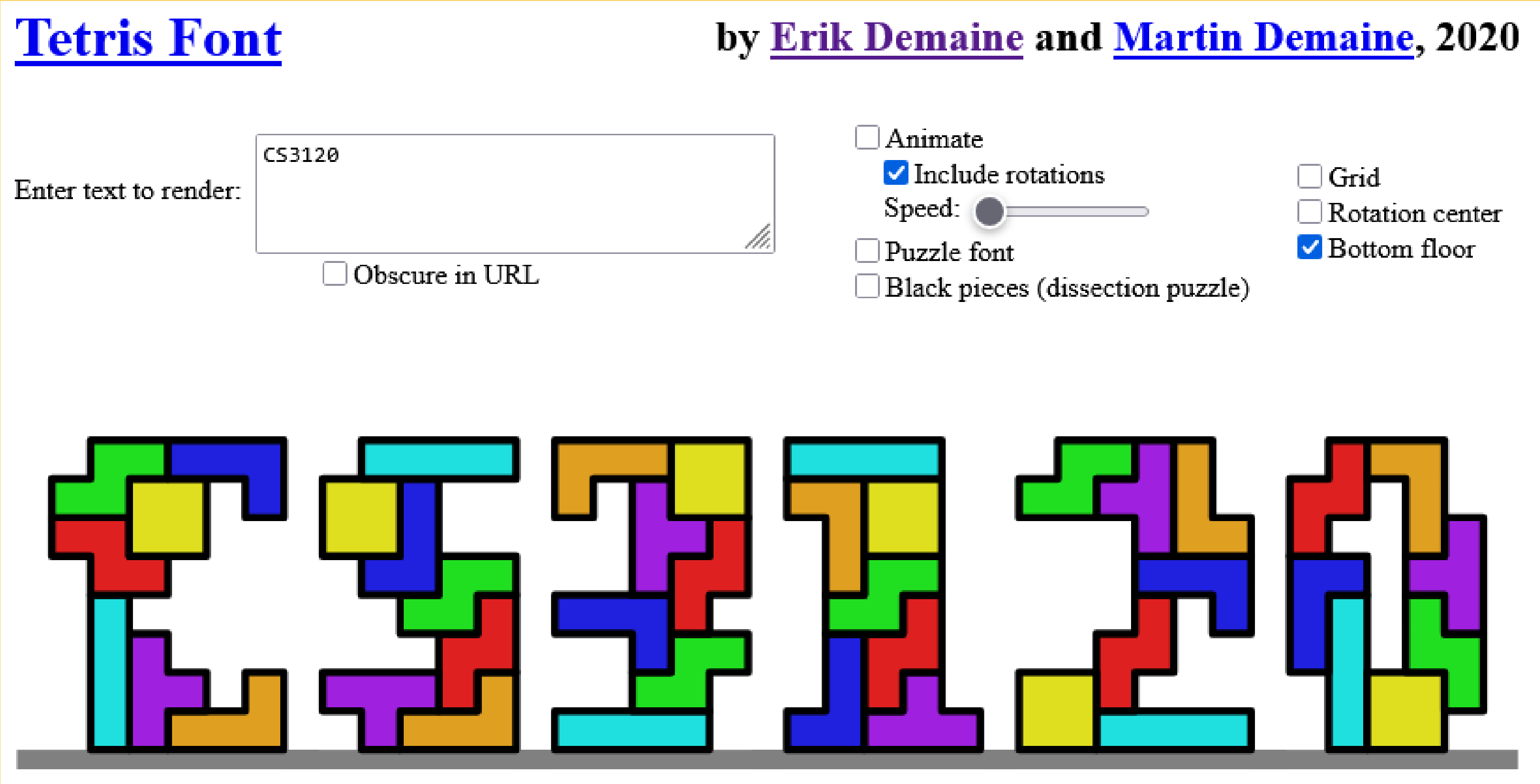
Final this Friday, May 2.



https://erikdemaine.org/fonts/tetris/?text=CS3120

Class 27:
Reductions Review

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Final: Friday, May 2, 2-4pm

- Everything is closed except for 1 page, double sided, letter size note.
- Same classroom, Rice 130
- Scope: mostly after midterm (may use some earlier stuff)
- Preparation: if you cleared all PS & PRR, you are good. Additional exercises: textbook, other CS3120s
- Relevant textbook sections will be updated soon: https://weikailin.github.io/cs3120-toc/outline/

Recap: Cook-Levin Theorem

Cook-Levin Theorem (Theorem 15.6 in the TCS Book):

For every $F \in \mathbf{NP}$, $F \leq_p$ **3SAT.**

Equivalently: **3SAT** is (in) **NP-Hard**

Recap NP-Complete

Definition: A function G is **NP-Hard** if every $F \in \mathbf{NP}$ can be reduced to G: $F \leq_P G$.

Definition: A function G is **NP-Complete** if $G \in \mathbf{NP}$ and G is **NP-Hard**.

Cook: **3SAT** is **NP-Hard**

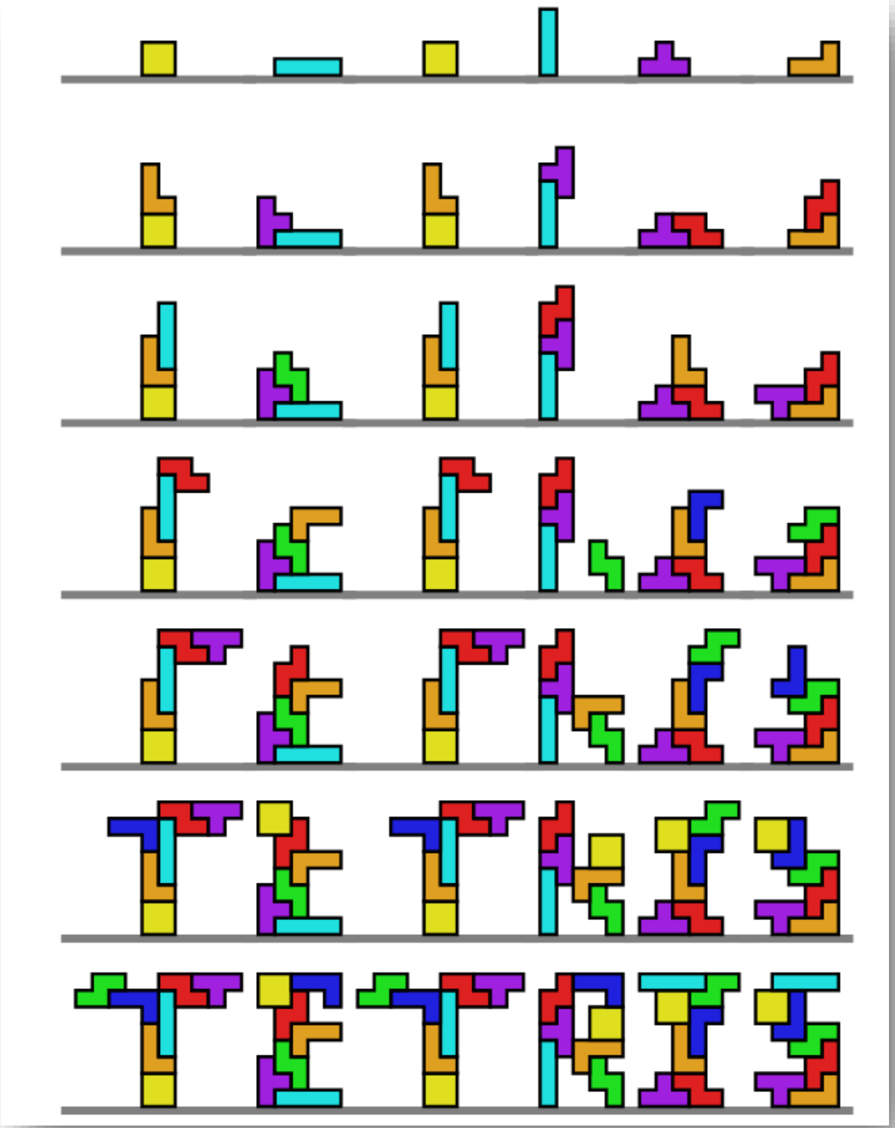
⇒ **3SAT** is **NP-Complete** by **3SAT** ∈ **NP**

Tetris is NP-Hard

Tetris is NP-hard even with $O(1)$ rows or columns

Sualeh Asif* Michael Coulombe* Erik D. Demaine* Martin L. Demaine*
Adam Hesterberg* Jayson Lynch* Mihir Singhal*

29 Sep 2020
https://arxiv.org/pdf/2009.14336



Bejeweled, Candy Crush (…) are NP-Hard

Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard

L. Gualà[1], S. Leucci[2], and E. Natale[3]

[1]Università degli Studi di Roma *Tor Vergata*
guala@mat.uniroma2.it
[2]Università degli Studi dell'Aquila
stefano.leucci@univaq.it
[3]*Sapienza* Università di Roma
natale@di.uniroma1.it

March 25, 2014

(a) Candy Crush Saga (b) Bejeweled (c) Shariki

https://arxiv.org/pdf/1403.5830

How to prove $\boldsymbol{F}$ is NP-Complete?

We have:

Definition: A function G is **NP-Hard** if every $F \in \mathbf{NP}$ can be reduced to G**:** $F \leq_P G$.

Definition: A function G is **NP-Complete** if $G \in \mathbf{NP}$ and G is **NP-Hard**.

Cook-Levin Theorem (Theorem 15.6 in the TCS Book):
For every $F \in \mathbf{NP}$, $F \leq_p$ **3SAT.**

We want to show:

- F is in NP:

A function $F: \{0,1\}^* \to \{0,1\}$ is in **NP** if there exists some $a \in \mathbb{N}^+$and $V: \{0,1\}^* \to \{0,1\}$ such that $V \in \mathbf{P}$ and $\forall x \in \{0,1\}^n, F(x) = 1 \leftrightarrow \exists w \in \{0,1\}^{n^a}$ such that $V(x,w) = 1$.

- F is NP-Hard:

Definition: A function G is **NP-Hard** if every $F \in \mathbf{NP}$ can be reduced to G: $F \leq_P G$.

Equivalently:

Definition: A function G is **NP-Hard** if **3SAT** $\leq_p G$.

Cook-Levin Theorem (Theorem 15.6 in the TCS Book):

For every $F \in \mathbf{NP}$, $F \leq_p$ **3SAT.**

Maximum Independent Set

Definition: For any undirected graph $G = (V, E)$, we say a set $S \subseteq V$ is an *independent set* if for all $u, v \in S$, the pair $(u, v) \notin E$ is not an edge.

For any undirected graph G and integer k, the Boolean function $ISET$ is defined to be $ISET$(G,k) = 1 iff G has an independent set of size at least k.

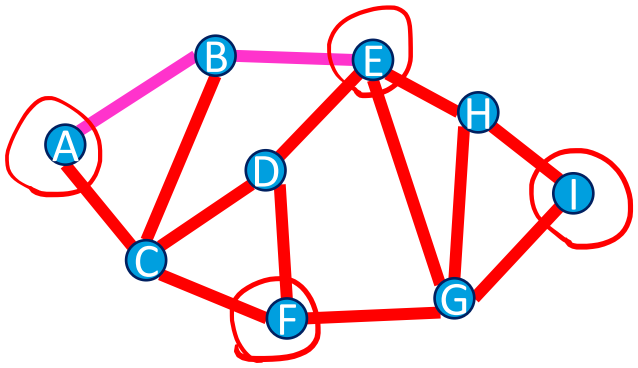
Example:

S1 = {A, I } ✓

S2 = {A, E, F, I } ✓

S3 = {B, D, G, H } ✗

$ISET(G, 3)$ = 1 $ISET(G, 6)$ = 0



Is ISET NP-Complete?

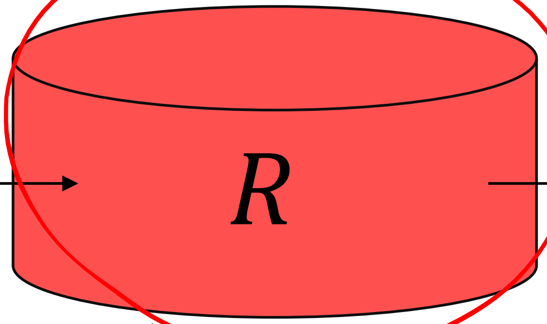
Max Independent Set:

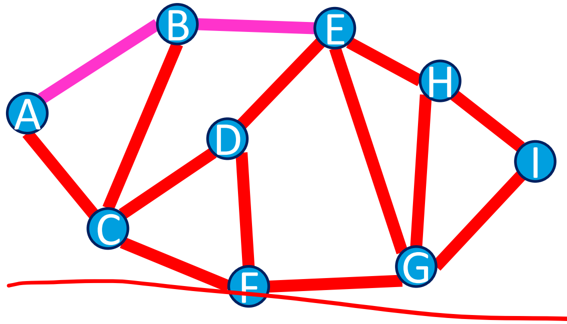
For any undirected graph G and integer k, the Boolean function $ISET$ is defined to be $ISET$(G,k) = 1 iff G has an independent set of size at least k.

- Is $ISET$ in NP?
- Is $ISET$ NP-Hard? **3SAT** $\leq_p$ $ISET$?
  What's the reduction?
  $R \in \mathbf{P}$ such that for all x, $3SAT(x) = ISET(R(x))$.

3 CNF

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



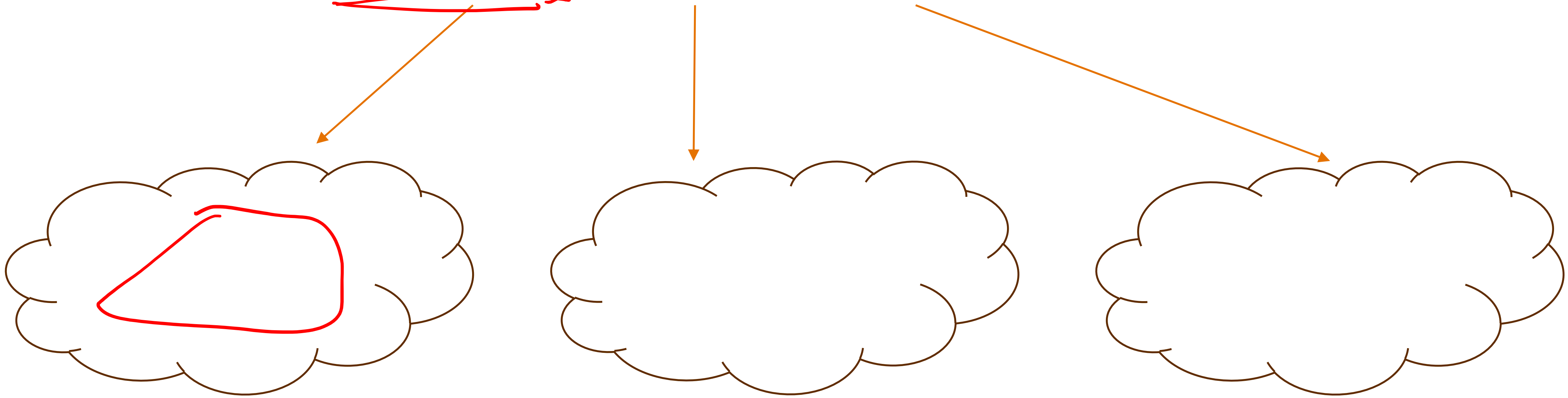


,k

How to reduce 3SAT to ISET?

Idea: to map each clause into a “gadget” satisfying some (desired) properties.

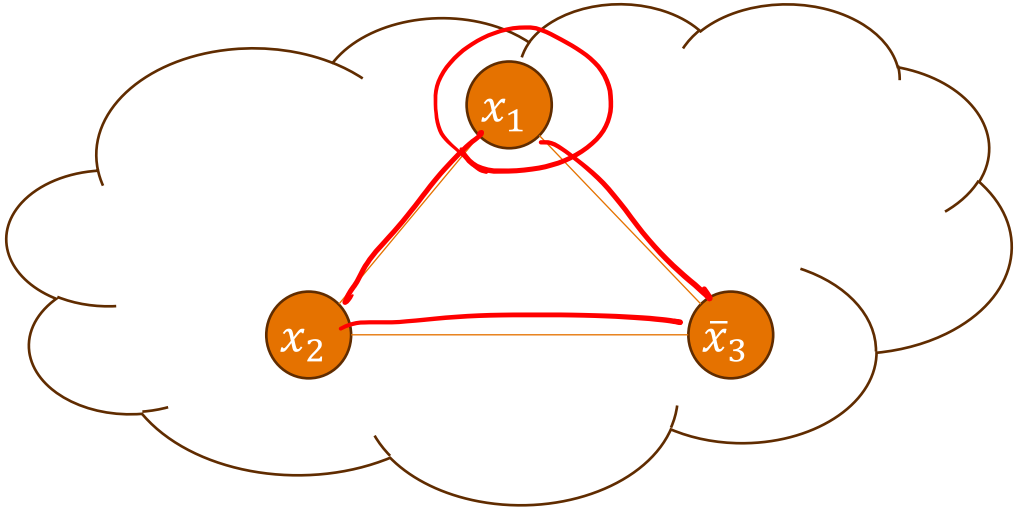
$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_2 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$



Gadget is triangle

$\underline{(x_1 \vee x_2 \vee \bar{x}_3)} \Longleftrightarrow 1$

any 1



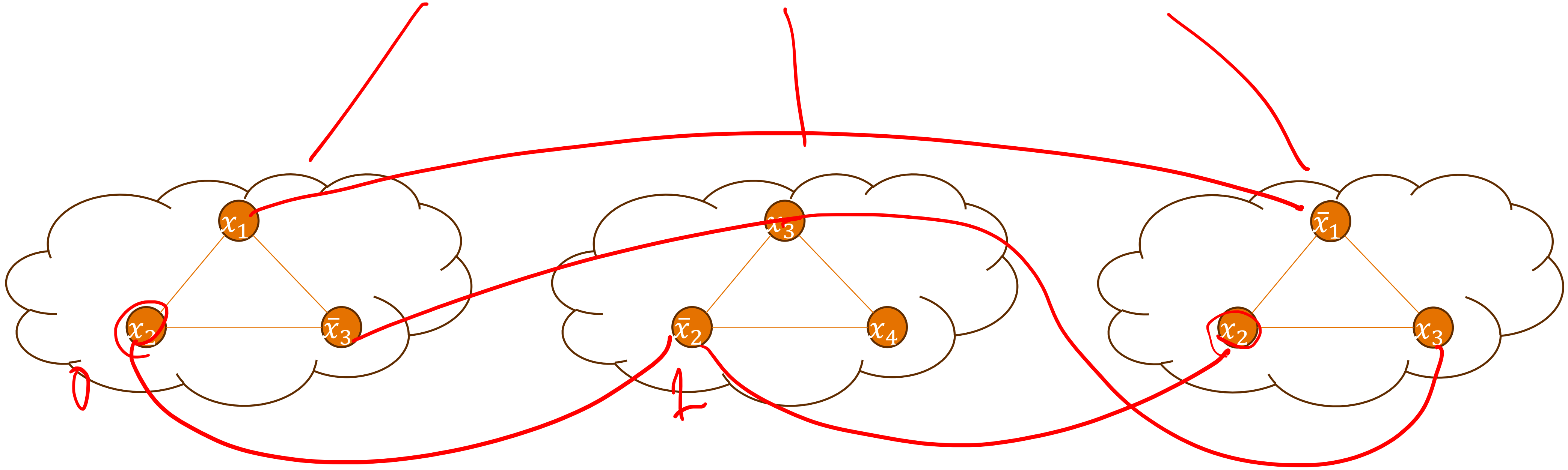
~~k = 1~~

Connect Gadgets

"Two literals in two clauses are negation of each other" is the global constraint

⇒ Put an edge as a constraint in ISET

$$(x_1 \lor x_2 \lor \bar{x}_3) \land (x_3 \lor \bar{x}_2 \lor x_4) \land (\bar{x}_1 \lor x_2 \lor x_3)$$



k = 3

Formalize the proof

Let x be a 3CNF, $x = c_1 \wedge c_2 \ldots \wedge c_n$ for some n.

For each $i \in [n]$, $c_i = a_{i1} \vee a_{i2} \vee a_{i3}$ for some literals a_{ij}.

Reduction $R(x)$ is constructed as follows.

$R(x)$: // input is a 3CNF instance $x = c_1 \wedge c_2 \cdots$

$G = (V, E)$ $V = \{ \quad \}$

$E = \{ \quad \}$

$n = n$

Output (G, n) // output is a ISET instance

Formalize the proof

Let x be a 3CNF, $x = c_1 \land c_2 \ldots \land c_n$ for some n.

For each $i \in [n]$, $c_i = a_{i1} \lor a_{i2} \lor a_{i3}$ for some literals a_{ij}.

Reduction $R(x)$ is constructed as follows.

$R(x)$:
1. Let $G = (V, E)$ be a graph of $3n$ vertices, $V = \{i1, i2, i3 : i \in [n]\}$.
2. The edges E is constructed as:
   a. For all $i \in [n]$, add edges $(i1, i2), (i2, i3), (i1, i3)$ to E triangles
   b. For all $i, j \in [n]$, if $i \neq j$ and $a_{is} = \bar{a}_{jt}$ for some $s, t \in \{1,2,3\}$, add edge (is, jt) to E
3. Output (G, n)

Complete the proof with analysis

Let x be a 3CNF, $x = c_1 \wedge c_2 \ldots \wedge c_n$ for some n.

For each $i \in [n]$, $c_i = a_{i1} \vee a_{i2} \vee a_{i3}$ for some literals a_{ij}.

Reduction $R(x)$ is constructed as follows.

$R(x)$:
1. Let $G = (V, E)$ be a graph of $3n$ vertices, $V = \{i1, i2, i3 : i \in [n]\}$.
2. The edges E is constructed as:
   a. For all $i \in [n]$, add edges $(i1, i2), (i2, i3), (i1, i3)$ to E
   b. For all $i, j \in [n]$, if $i \neq j$ and $a_{is} = \bar{a}_{jt}$ for some $s, t \in \{1,2,3\}$, add edge (is, jt) to E
3. Output (G, n)

Analysis:

want 3SAT(x) = ISET($R(x)$) and $R \in \mathbf{P}$.

3SAT(x) = 1 iff . . . . . iff ISET(R(x)) = 1

Complete the proof with analysis

Let x be a 3CNF, $x = c_1 \wedge c_2 \dots \wedge c_n$ for some n.

For each $i \in [n]$, $c_i = a_{i1} \vee a_{i2} \vee a_{i3}$ for some literals a_{ij}.

Reduction $R(x)$ is constructed as follows.

$R(x)$:
1. Let $G = (V, E)$ be a graph of $3n$ vertices, $V = \{i1, i2, i3 : i \in [n]\}$.
2. The edges E is constructed as:
   a. For all $i \in [n]$, add edges $(i1, i2), (i2, i3), (i1, i3)$ to E
   b. For all $i, j \in [n]$, if $i \neq j$ and $a_{is} = \bar{a}_{jt}$ for some $s, t \in \{1,2,3\}$, add edge (is, jt) to E
3. Output (G, n)

Analysis:
If x is satisfiable, 3SAT(x) = 1

def 3SAT

$\Leftrightarrow$ Exists assignment such that for each i, a_{is} = 1 for some s.

by $R(x)$

$\Leftrightarrow$ $U = \{is : a_{is} = 1, i \in [n]\}$ is an indep set of size n for R(x) bcs for all j, t s.t. $a_{is} = \bar{a}_{jt}$, jt is not in U.

def ISET

$\Leftrightarrow$ ISET($R(x)$) = 1.

Hence, 3SAT(x) = (ISET($R(x)$) for all x. Also, we have $R \in \mathbf{P}$. Thus, 3SAT$\leq_p$ISET.

Framework of proofs

Precondition / notation

Let x be a 3CNF, $x = c_1 \wedge c_2 \ldots \wedge c_n$ for some n.

For each $i \in [n]$, $c_i = a_{i1} \vee a_{i2} \vee a_{i3}$ for some literals a_{ij}.

Reduction $R(x)$ is constructed as follows.

Algorithm of reduction

$R(x)$:

1. Let $G = (V, E)$ be a graph of $3n$ vertices, $V = \{i1, i2, i3 : i \in [n]\}$.
2. The edges E is constructed as:
   a. For all $i \in [n]$, add edges $(i1, i2), (i2, i3), (i1, i3)$ to E
   b. For all $i, j \in [n]$, if $i \neq j$ and $a_{is} = \bar{a}_{jt}$ for some $s, t \in \{1,2,3\}$, add edge (is, jt) to E
3. Output (G, n)

Analysis:

If x is satisfiable, 3SAT(x) = 1

⇔ Exists assignment such that for each i, a_{is} = 1 for some s.

⇔ $U = \{is : a_{is} = 1, i \in [n]\}$ is an indep set of size n for R(x) bcs for all j, t s.t. $a_{is} = \bar{a}_{jt}$, jt is not in U.

⇔ ISET($R(x)$) = 1.

Hence, 3SAT(x) = (ISET($R(x)$) for all x. Also, we have $R \in \mathbf{P}$. Thus, 3SAT$\leq_p$ISET.

Analysis of reduction

Review of this course

Big idea: Counting and Diagonalizing

- Uncountable sets: Boolean functions, real numbers, …
- Countable sets: integers, strings

- Uncomputable functions
- Computable functions: Turing machines, DFAs, …

- All are infinite! Counting is way to go through them.

Class 5

Are they the same (or comparable)?

easier

$$
\begin{aligned}
s_1 &= 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\ldots\\
s_2 &= 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\ldots\\
s_3 &= 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\ldots\\
s_4 &= 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\ldots\\
s_5 &= 1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\ldots\\
s_6 &= 0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\ldots\\
s_7 &= 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\ldots\\
s_8 &= 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\ldots\\
s_9 &= 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\ldots\\
s_{10} &= 1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\ldots\\
s_{11} &= 1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\ldots\\
\vdots\ \ & \quad \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots
\end{aligned}
$$

$$s = 1\,0\,1\,1\,1\,0\,1\,0\,0\,1\,1\ldots$$

Proof. For all sets S, $|pow(S)| > |S|$.

Towards a contradiction, **assume** $\exists \boldsymbol{S}.\ |\boldsymbol{pow}(\boldsymbol{S})| \leq |\boldsymbol{S}|$.
By the definition of $\leq$, there must exist a *surjective function* g from $S \to pow(S)$.

Define $T = \{\, a \mid a \notin g(a), a \in S \,\}$. ??

$T \in pow(S)$. (Obviously, its a subset of S.)
Since g is surjective, $\exists\ u \in S$ such that $g(u) = T$.

(1) If $u \in g(u)$, then $u \notin T$.
But $T = g(u)$, so $u \notin g(u)$.

(2) If $u \notin g(u)$, then $u \in T$.
But $T = g(u)$, so $u \in g(u)$.

Contradiction! So, there must not exist any S such that $|pow(S)| \leq |S|$.

3

5

Big idea: Reductions

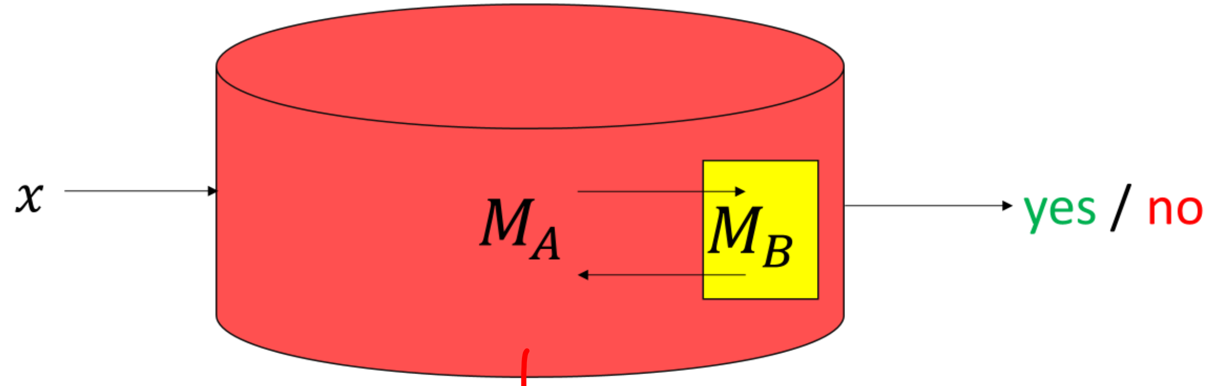
- Reduction from problem A to problem B:
  Solving (the instances of) A by
  solving (the instances of) B
- Show A is solvable if B is solvable (algorithms, DFA = RE)
- Show B is hard if A is hard (uncomputability, NP hardness)

Class 22

Another way to imagine reduction

Reducing “task” A to “task” B, denoted by $A \leq_R B$

1. Assume that algorithm M_B solves B
2. Design algorithm M_A (that uses M_B as subroutine)



Reduction

Which is easier, A or B?

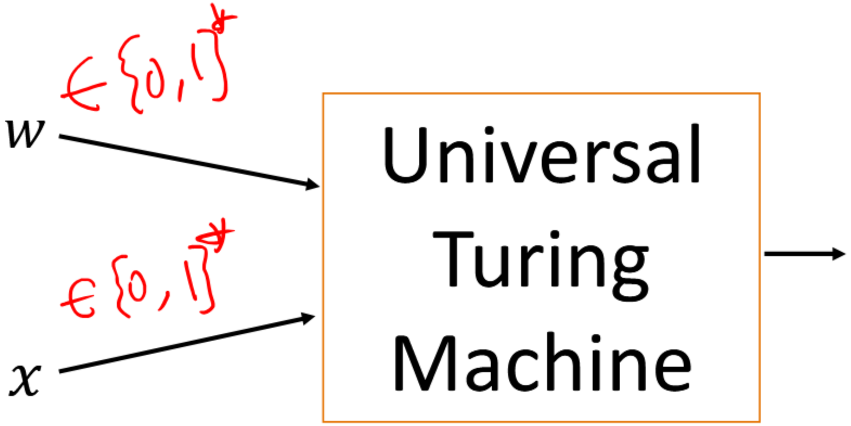
B may “look” easier!

Big idea: Code as Data, Data as Code

- Natural
- A higher order: questions / functions on code
- Manipulate data / simulate code in reductions
- Tool: universal circuit, universal Turing machine

Class 19

Is there a “Universal Turing Machine”?



$$\begin{cases} TM_w(x) & \text{if } w \text{ represents TM} \\ 0 & \text{otherwise} \end{cases}$$

We can simulate any Turing machine using our favorite programming lang (assume unbounded time & space).

Gödel’s Incompleteness Theorem

What is a proof?

"Take any definite unsolved problem, such as …. However unapproachable these problems may seem to us and however helpless we stand before them, we have, nevertheless, the firm conviction that their solution must follow by a finite number of purely logical processes…", David Hilbert, 1900.

local comp

Mathematical statements

A mathematical statement is simply a piece of text, binary string $x \in \{0,1\}^*$

"*The number* $2{,}696{,}635{,}869{,}504{,}783{,}333{,}238{,}805{,}675{,}613{,}\ 588{,}278{,}597{,}832{,}162{,}617{,}892{,}474{,}670{,}798{,}113$ *is prime*".

The following Python function halts on every positive integer `n`

```
def f(n):
    if n==1: return 1
    return f(3*n+1) if n % 2 else f(n//2)
```

Big idea: A *proof* is just a string of text whose meaning is given by a *verification algorithm*.

Definition 11.2 (Proof systems)

Let $\mathcal{T} \subseteq \{0,1\}^*$ be some set (which we consider the "true" statements). A *proof system* for $\mathcal{T}$ is an algorithm V that satisfies:

1. *(Effectiveness)* For every $x, w \in \{0,1\}^*$, $V(x,w)$ halts with an output of either 0 or 1.
2. *(Soundness)* For every $x \notin \mathcal{T}$ and $w \in \{0,1\}^*$, $V(x,w) = 0$.

Trivial: V(x,w) = 0 for all x,w (not useful).

Big idea: A *proof* is just a string of text whose meaning is given by a *verification algorithm*.

T = { "2 is prime" }

Definition 11.2 (Proof systems)

Let $\mathcal{T} \subseteq \{0,1\}^*$ be some set (which we consider the "true" statements). A *proof system* for $\mathcal{T}$ is an algorithm V that satisfies:

1. *(Effectiveness)* For every $x, w \in \{0,1\}^*$, $V(x,w)$ halts with an output of either 0 or 1.
2. *(Soundness)* For every $x \notin \mathcal{T}$ and $w \in \{0,1\}^*$, $V(x,w) = 0$.

A true statement $x \in \mathcal{T}$ is *unprovable* (with respect to V) if for every $w \in \{0,1\}^*$, $V(x,w) = 0$. We say that V is *complete* if there does not exist a true statement x that is unprovable with respect to V.

$x \in T$ is *provable* if exists w such that $V(x,w) = 1$.
V is *complete* if for all $x \in T$, x is provable.

Gödel's Incompleteness Theorem

Theorem 11.3 (Gödel's Incompleteness Theorem: computational variant)

There does not exist a complete proof system for $\mathcal{H}$.

Turing machines,
NotHaltOnZero

There is a set of true statements H, but:
H is incomplete for all prove system V

$x \in H$ $TM_V(x, w) = 0$ all w

$\bot$ not halt

Some V can prove that $x \in H$ for some x.
But exists $x' \in H$ such that the same V can not prove.

Reduction from Halting to Incompleteness

Algorithm 11.4 Halting from proofs

Input: Turing machine M

Output: 1 M if halts on input 0; 0 otherwise.

for$\{n = 1, 2, 3, \ldots\}$

  for$\{w \in \{0,1\}^n\}$

    if$\{V(\text{"}M \text{ does not halt on } 0\text{"}, w) = 1\}$

      return 0

    endif

    Simulate M for n steps on 0.

    if$\{M$ halts$\}$

      return 1

    endif

  endfor

endfor

x eH ?

always halts by Eff of V

Assume V complete

soundness of V

$\nexists$ w

Case 1, M halts in T steps:
V always 0, Algo never return 0.
Algo return 1 when $n = T$.

Case 2, M never halt:
Algo never return 1.
By V is complete, exists w of N bits such that $V(\ldots, w) = 1$.
When $n = N$, Algo return 0.

Student Experiences of Teaching

https://in.virginia.edu/CourseXperience