Final Exam Topics 1

- Regular languages
  - Decision Problems
    - Membership
    - Emptiness
    - Finiteness
    - Σ^*
    - Equality
    - Containment
  - Closure
    - Union/Concatenation/Star
    - Complement
    - Substitution/Quotient, Prefix, Infix, Suffix
    - Max/Min

Regular language decision problems

- Every regular language can be recognized by a DFA
- Membership of a string, w, in a DFA's language can be determined by running the machine for |w| steps and seeing if it accepts or rejects
- Every DFA can be reduced to a unique minimum state DFA
- We can easily recognize if a minimized DFA accepts finite or infinite languages, or even empty or Σ*
- We can determine equality of languages by equality of minimized DFAs
- We can determine containment of B in A by intersecting them and seeing the resulting reduced machine is the same as that which accepted A.
- Regular languages are closed under union, concatenation, star, intersection, complement, substitution, max and min

Final Exam Topics 2

- Context free languages
    - Writing a simple CFG
    - Decision Problems
        - Membership
        - Emptiness
        - Finiteness
        - Σ^* (undecidable)
        - Equality (undecidable)
        - Containment (undecidable)
    - Closure
        - Union/Concatenation/Star
        - Intersection with Regular
        - Substitution/Quotient with Regular, Prefix, Infix, Suffix
    - Non-closure
        - intersection, complement, quotient, Max/Min
    - Pumping Lemma for CFLs

Final Exam Topics 3

- Chomsky Hierarchy
  (Red involve no constructive questions)
  - Regular, CFG, CSG, PSG (type 3 to type 0)
  - FSAs, PDAs, LBAs, Turing machines
  - Length preservation or increase makes membership in associated languages decidable for all but PSGs
  - CFLs can be inherently ambiguous but that does not mean a language that has an ambiguous grammar is automatically inherently ambiguous

Context free language decision problems

- Every CFL can be recognized by a PDA
- Membership of a string, w, where |w|=n, in a CFL can be determined by using the $O(n^3)$ CKY algorithm
- There is no way to find a unique minimum PDA but you can reduce a CFG to one that has has no useless rules or non-terminals. With this you can test for emptiness
- There is no algorithm to determine if a CFG generates Σ^*. However, by reducing a grammar we can test for emptiness, finiteness and infiniteness.
- Equality of CFLs is also unsolvable; can just ask if the CFG produces Σ^*
- CFLs are closed under union, concatenation, star, substitution, and intersection with regular but not under complement, intersection, min or max

Final Exam Topics 4

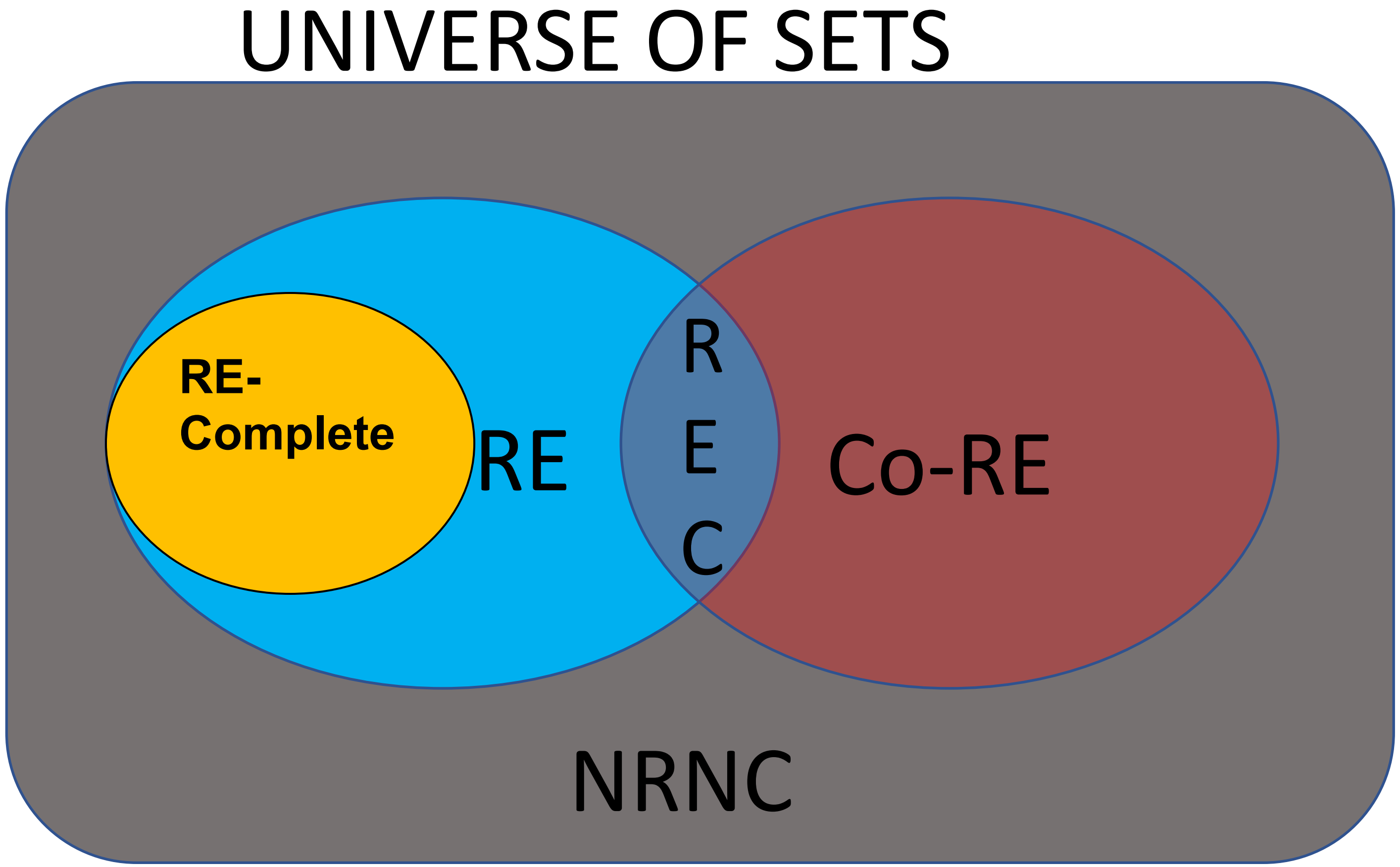
- Computability Theory
  - Decision problems: solvable (decidable, recursive), semi-decidable (recognizable, recursively enumerable/re, generable), non-re
  - A set is re iff it is semi-decidable
  - If set is re and complement is also re, set is recursive (decidable)
  - Halting problem (K_0): diagonalization proof of undecidability
    - Set K_0 is re but complement is not
  - Set K = { f | f(f) converges }
  - Algorithms (Total): diagonalization proof of non-re
  - Reducibility to show certain problems are not decidable or even non-re
  - K and K_0 are re-complete – reducibility to show these results
  - Rice's Theorem: All non-trivial I/O properties of functions are undecidable (weak and strong versions)
  - Use of quantification to discover upper bound on complexity

Final Exam Topics 5

- Computability Applied to Formal Grammars
  (Red only results not constructions that lead to these)
  - Post Correspondence problem (PCP)
    - Definition
    - Undecidability (proof was only sketched and is not part of this course)
    - Application to ambiguity and non-emptiness of intersections of CFLs and to non-emptiness of CSLs
  - Traces of Turing computations
    - Not CFLs
    - Single steps are CFLs (use reversal of second configuration)
    - Intersections of pairwise correct traces are traces
    - Complement of traces (including terminating traces) are CFL
    - Use to show cannot decide if CFL, L, is Σ^*
    - $L = \Sigma^*$ and $L = L^2$ are undecidable for CFLs
  - PSG can mimic TM, so can generate any re language; thus, membership in PSL is undecidable, as is emptiness of PSL.
  - All re sets are homomorphic images of CSLs (erase fill character)

Final Exam Topics 6

- Complexity Theory
  - Verifiers versus solvers: P versus NP
  - Definitions of NP: verify in deterministic poly time vs solve in non-deterministic polynomial time
  - Co-P and co-NP; NP-Hard versus NP-Complete
  - Basic idea behind SAT as NP-Complete
  - Reduction of SAT to 3-SAT to Subset-Sum
  - Equivalence of Subset-Sum to Partition
  - Relation of Subset-Sum and Partition to multiprocessor scheduling
  - Vertex cover, 3-coloring, register allocation, Independent set, 0-1 integer linear programming
  - Gadgets for above



NR (non-recursive)

= (NRNC ∪ Co-RE) - REC

1. Let set **A** be recursive, **B** be re non-recursive and **C** be non-re. Choosing from among **(REC) recursive**, **(RE) re non-recursive**, **(NR) non-re**, categorize the set **D** in each of a) through d) by listing **all** possible categories. No justification is required.

a.) $\mathbf{D = \sim C}$ **RE, NR**

b.) $\mathbf{D \subseteq (A \cup C)}$ **REC, RE, NR**

c.) $\mathbf{D = \sim B}$ **NR**

d.) $\mathbf{D = B - A}$ **REC, RE**

2. Prove that the **Halting Problem** (the set $\mathbf{K_0}$) is not recursive (decidable) within any formal model of computation. (Hint: A diagonalization proof is required here.)

Assume we can decide the halting problem. Then there exists some total function Halt such that

$$\text{Halt}(x,y) = \begin{cases} 1 & \text{if } [x]\ (y) \text{ is defined} \\ 0 & \text{if } [x]\ (y) \text{ is not defined} \end{cases}$$

Here, we have numbered all programs and [x] refers to the x-th program in this ordering. We can view Halt as a mapping from $\aleph$ into $\aleph$ by treating its input as a single number representing the pairing of two numbers via the one-one onto function

$$\text{pair}(x,y) = \langle x,y \rangle = 2^x\ (2y + 1) - 1$$

with inverses $\langle z \rangle_1 = \exp(z+1,1)$ **and** $\langle z \rangle_2 = (((z + 1)\ //\ 2^{\langle z \rangle 1}\ ) - 1)\ //\ 2$

Now if Halt exist, then so does Disagree, where

$$\text{Disagree}(x) = \begin{cases} 0 & \text{if Halt}(x,x) = 0\text{, i.e, if } \Phi_x\ (x) \text{ is not defined} \\ \mu y\ (y == y+1) & \text{if Halt}(x,x) = 1\text{, i.e, if } \Phi_x\ (x) \text{ is defined} \end{cases}$$

Since Disagree is a program from $\aleph$ into $\aleph$, Disagree can be reasoned about by Halt. Let d be such that Disagree = Φ_d, then

$$\text{Disagree}(d) \text{ is defined} \Leftrightarrow \text{Halt}(d,d) = 0 \Leftrightarrow \Phi_d\ (d) \text{ is undefined} \Leftrightarrow \text{Disagree}(d) \text{ is undefined}$$

But this means that Disagree contradicts its own existence. Since every step we took was constructive, except for the original assumption, we must presume that the original assumption was in error. Thus, the Halting Problem is not solvable.

3.Using reduction from the known undecidable **HasZero,**
HZ = { f | $\exists$x f(x) = 0 }, show the non-recursiveness (undecidability) of the problem to decide if an arbitrary recursive function **g** has the property **IsZero, Z = { f | $\forall$x f(x) = 0 }**.

HZ = { f | $\exists$x $\exists$t [STP(f, x, t) & VALUE(f, x, t) == 0] }

Let f be the index of an arbitrary effective procedure.

Define g_f(y) = 1 - $\exists$x $\exists$t [STP(f, x, t) & VALUE(f, x, t) == 0]

If $\exists$x f(x) = 0, we will find the x and the run-time t, and so we will return 0 (1 – 1)

If $\forall$x f(x) $\neq$ 0, then we will diverge in the search process and never return a value.

Thus, f $\in$ HZ iff g_f $\in$ Z = { f | $\forall$x f(x) = 0 }.

4. Choosing from among **(D) decidable**, **(U) undecidable**, **(?) unknown**, categorize each of the following decision problems. No proofs are required.

Problem / Language Class	**Regular**	**Context Free**
$L = \Sigma^*$?	*D*	*U*
$L = \phi$?	*D*	*D*
$x \in L^2$, for arbitrary x ?	*D*	*D*

5. Choosing from among **(Y) yes, (N) No, (?) unknown**, categorize each of the following closure properties. No proofs are required.

Problem / Language Class	**Regular**	**Context Free**
Closed under intersection?	***Y***	***N***
Closed under quotient?	***Y***	***N***
Closed under quotient with Regular languages?	***Y***	***Y***
Closed under complement?	***Y***	***N***

6. Prove that any class of languages, ***C***, closed under union, concatenation, intersection with regular languages, homomorphism and substitution (e.g., the Context-Free Languages) is closed under **MissingMiddle**, where, assuming L is over the alphabet Σ,
MissingMiddle(L) = $\{ xz \mid \exists y \in \Sigma^* \text{ such that } xyz \in L \}$
You must be very explicit, describing what is produced by each transformation you apply.

Define the alphabet $\Sigma' = \{ a' \mid a \in \Sigma \}$, where, of course, a' is a "new" symbol, i.e., one not in Σ.

Define homomorphisms g and h, and substitution f as follows:
$g(a) = a' \quad \forall a \in \Sigma$ $\qquad h(a) = a \;;\; h(a') = \lambda \quad \forall a \in \Sigma$ $\qquad f(a) = \{a, a'\} \quad \forall a \in \Sigma$

Consider $R = \Sigma^* \bullet g(\Sigma^*) \bullet \Sigma^* = \{ x\, y'\, z \mid x, y, z \in \Sigma^* \text{ and } y' = g(y) \in \Sigma'^* \}$
Σ^* is regular since it is the Kleene star closure of a finite set.
$g(\Sigma^*)$ is regular since it is the homomorphic image of a regular language.
R is regular as it is the concatenation of regular languages.

Now, $f(L) = \{ f(w) \mid w \in L \}$ is in C since C is closed under substitution. This language is the set of strings in L with randomly selected letters primed. Any string $w \in L$ gives rise to $2^{|w|}$ strings in f(L).

$f(L) \cap R = \{ x\, y'\, z \mid x\, y\, z \in L \text{ and } y' = g(y) \}$ is in C since C is closed under intersection with regular languages.

MissingMiddle(L) = $h(\, f(L) \cap R\,) = \{ x\, z \mid \exists y \in \Sigma^* \text{ such that } xyz \in L \}$ which is in C, since C is closed under homomorphism. Q.E.D.

7. Use **PCP** to show the undecidability of the problem to determine if the intersection of two context free languages is non-empty. That is, show how to create two grammars $\mathbf{G_A}$ and $\mathbf{G_B}$ based on some instance $\mathbf{P} = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle$ of **PCP**, such that $\mathbf{L(G_A) \cap L(G_B) \neq \phi}$ iff **P** has a solution. Assume that **P** is over the alphabet $\mathbf{\Sigma}$. You should discuss what languages your grammars produce and why this is relevant, but no formal proof is required.

$G_A = (\{ A \} , \Sigma \cup \{ [\, i\,] \mid 1 \leq i \leq n \} , A , P_A \}$

$P_A : A \rightarrow x_i\, A\, [\, i\,] \mid x_i\, [\, i\,]$

$G_B = (\{ B \} , \Sigma \cup \{ [\, i\,] \mid 1 \leq i \leq n \} , B , P_B \}$

$P_B : A \rightarrow y_i\, B\, [\, i\,] \mid y_i\, [\, i\,]$

$L(G_A) = \{ x_{i1}\ x_{i2} \ldots x_{ip}\ [i_p] \ldots [i_2]\ [i_1] \mid p \geq 1, 1 \leq i_t \leq n, 1 \leq t \leq p \}$

$L(G_B) = \{ y_{j1}\ y_{j2} \ldots y_{jq}\ [j_q] \ldots [j_2]\ [j_1] \mid q \geq 1, 1 \leq j_u \leq n, 1 \leq u \leq q \}$

$L(G_A) \cap L(G_B) = \{ w\ [k_r] \ldots [k_2]\ [k_1] \mid r \geq 1, 1 \leq k_v \leq n, 1 \leq v \leq r \}$, where

$$w = x_{k1}\, x_{k2} \ldots x_{kr} = y_{k1}\, y_{k2} \ldots y_{kr}$$

If $L(G_A) \cap L(G_B) \neq \phi$ then such a w exists and thus $k_1 , k_2 , \ldots , k_r$ is a solution to this instance of PCP. This shows that a decision procedure for the non-emptiness of the intersection of CFLs implies a decision procedure for PCP, which we have already shown is undecidable. Hence, the non-emptiness of the intersection of CFLs is undecidable. Q.E.D.

8. Consider the set of indices **CONSTANT** = { **f** | ∃**K** ∀**y** [$\varphi_f(y)$ = **K**] }. Use Rice's Theorem to show that **CONSTANT** is not recursive. Hint: There are two properties that must be demonstrated.

First, show CONSTANT is non-trivial.

Z(x) = 0 is in CONSTANT

S(x) = x+1 is not in CONSTANT

Thus, CONSTANT is non-trivial

Second, let f, g be two arbitrary computable functions with the same I/O behavior.

That is, ∀x, if f(x) is defined, then f(x) = g(x); otherwise both diverge, i.e., f(x)↑ and g(x)↑

Now, f ∈ CONSTANT

⇔ ∃K ∀x [f(x) = K]	**by the definition of CONSTANT**
⇔ ∀x [g(x) = C]	**where C is the instance of K above, since ∀x [f(x) = g(x)]**
⇔ ∃K ∀x [g(x) = K]	**from above**
⇔ g ∈ CONSTANT	**by the definition of CONSTANT**

Since CONSTANT meets both conditions of Rice's Theorem, it is undecidable. Q.E.D.

9. Show that **CONSTANT** $\equiv_m$ **TOT**, where **TOT** = { f | $\forall$y $\varphi_f(y)\downarrow$ }.

CONSTANT $\leq_m$ TOT

Let f be an arbitrary effective procedure.

Define g_f by

$g_f(0) = f(0)$

$g_f(y+1) = f(y+1) + \mu z\ [f(y+1) = f(y)]$

Now, if f $\in$ CONSTANT then $\forall$y [f(y)$\downarrow$ and [f(y+1) = f(y)]].

Under this circumstance, μz [f(y+1) = f(y)] is 0 for all y and $g_f(y) = f(y)$ for all y.

Clearly, then $g_f \in$ TOT

If, however, f $\notin$ CONSTANT then $\exists$y [f(y+1) $\neq$ f(y)] or $\exists$y f(y)$\uparrow$.

Choose the least y meeting this condition.

If f(y)$\uparrow$ then $g_f(y)\uparrow$ since f(y) is in $g_f(y)$'s definition (the 1st term).

If f(y)$\downarrow$ but [f(y+1) $\neq$ f(y)] then $g_f(y)\uparrow$ since μz [f(y+1) = f(y)]$\uparrow$ (the 2nd term).

Clearly, then $g_f \notin$ TOT

Combining these, f $\in$ CONSTANT $\Leftrightarrow$ $g_f \in$ TOT and thus CONSTANT $\leq_m$ TOT

$TOT \le_m CONSTANT$

Let f be an arbitrary effective procedure.

Define g_f by $g_f(y) = f(y) - f(y)$

Now, if $f \in TOT$ then $\forall y\ [\ f(y){\downarrow}\ ]$ and thus $\forall y\ g_f(y) = 0$.

Clearly, then $g_f \in CONSTANT$

If, however, $f \notin TOT$ then $\exists y\ [f(y){\uparrow}\ ]$ and thus, $\exists y\ [g_f(y){\uparrow}]$. Clearly, then $g_f \notin CONSTANT$

Combining these, $f \in TOT \Leftrightarrow g_f \in CONSTANT$ and thus $TOT \le_m CONSTANT$

Hence, $CONSTANT \equiv_m TOT$. Q.E.D.

10. Why does Rice's Theorem have nothing to say about each of the following? Explain by showing some condition of Rice's Theorem that is not met by the stated property.

***a.*)** **AT-LEAST-LINEAR = { f | $\forall$y φ_f(y) converges in no fewer than y steps }.**

We can deny the 2nd condition of Rice's Theorem since

Z, where Z(x) = 0, implemented by the TM R converges in one step no matter what x is and hence is not in AT-LEAST-LINEAR

Z', defined by TM $\mathcal{L}\ \mathcal{R}$ R, is in AT-LEAST-LINEAR since it takes over 2*|input| steps.

However, $\forall$x [Z(x) = Z'(x)], so they have the same I/O behavior and yet one is in and the other is out of AT-LEAST-LINEAR, denying the 2nd condition of Rice's Theorem

***b.*)** **HAS-IMPOSTER = { f | $\exists$ g [g$\neq$f** and **$\forall$y [φ_g(y) = φ_f(y)]] }.**

We can deny the 1st condition of Rice's Theorem since all functions have an imposter. To see this, consider, for any function f, the equivalent but distinct function g(x) = f(x) + 0. Thus, HAS-IMPOSTER is trivial since it is equal to $\aleph$, the set of all indices.

13. Show a first-fit schedule for the following task times on two processors
{T1/1, T2/7, T3/2, T4/4, T5/4, T6/2, T7/5, T8/2, T9/3, T10/4}

T1	T3	T3	T4	T4	T4	T4	T5	T5	T5	T5	T8	T8	T9	T9	T9		
T2	T2	T2	T2	T2	T2	T2	T6	T6	T7	T7	T7	T7	T7	T10	T10	T10	T10

Sample Question#1

1. Given that the predicate **STP** and the function **VALUE** are algorithms, show that we can semi-decide

   HZ = { f | φ_f evaluates to 0 for some input}

   Note: **STP(f, x, s)** is true iff **φ_f(x)** converges in **s** or fewer steps and, if so, **VALUE(f, x, s) = φ_f(x)**.

f $\in$HZ iff $\exists$<x,t> [STP(f,x,t) & VALUE(F,x,t)=0] provides a semi-decision procedure

Sample Questions#2

2. Use Rice's Theorem to show that **HZ** is undecidable, where
   HZ = { f | φ_f evaluates to 0 for some input}

HZ is non-trivial as Zero(x) = 0 $\in$ HZ and S(x)= x+1 $\notin$ HZ

Let f, g be two arbitrary indices such that Range(f) = Range(g)

f $\in$ HZ	**iff 0 $\in$ Range(f)**	**definition of HZ**
	Iff 0 $\in$ Range(g)	**as ranges are the same**
	iff g $\in$ HZ	

Thus, HZ undecidable using oen of Rice's weaker versions

Sample Questions#3

3. Use Reduction from Halt to show that **HZ** is undecidable, where
   HZ = { f | φ_f evaluates to 0 for some input}

Let <f,x> be an arbitrary index and input

Define $\forall y\ g_{f,x}(y) = 1 - \exists \langle x,t \rangle$ [STP(f,x,t) & VALUE(F,x,t)=0]

$\langle f,x \rangle \in$ HALT iff $\forall y\ g_{f,x}(y) = 0$; otherwise $\forall y\ g_{f,x}(y) \uparrow$

Thus, $\langle f,x \rangle \in$ HALT iff $g_{f,x} \in$ HZ

As HALT ≤ HZ, HZ must be undecidable. Since it's RE, it is also RE-Complete.

Sample Question#4

4. Let **P = { f | ∃ x [STP(f, x, x)] }**. Why does Rice's theorem not tell us anything about the undecidability of **P**?

It is easy to show two functions, one of which operates in linear time (or even constant time) and the other in twice linear time, yet both compute the same function. A simple example is a TM that computes the constant Zero.

R takes one unit of time, independent of x, to compute 0.

$\mathcal{L}\,\mathcal{R}$R takes a bit over 2x time (really 2x+3) for all values of x.

Both compute Zero.

Sample Question#5

5. Let **S** be an re (recursively enumerable), non-recursive set, and **T** be an re, possibly recursive set. Let

E = { z | z = x + y, where x ∈ S and y ∈ T }.

Answer with proofs, algorithms or counterexamples, as appropriate, each of the following questions:

(a) Can **E** be non re? **No. If T = ∅ then E is recursive. Assume S is non-empty and S and T are enumerated by f_S, f_T, resp. Then $f_E(<x,y>) = f_S(x) + f_S(y)$ enumerates E.**

(b) Can **E** be re non-recursive? **Yes. T = {0}, E = S**

(c) Can **E** be recursive? **Yes, $T=\aleph$, E = { x | x ≥ min value in S }**

Some Quantification Examples

- $<f,x> \in \text{Halt} \Leftrightarrow \exists t\,[\ \text{STP}(f,x,t)\ ]$ RE
- $f \in \text{Total} \Leftrightarrow \forall x \exists t\,[\ \text{STP}(f,x,t)\ ]$ NRNC
- $f \in \text{NotTotal} \Leftrightarrow \exists x \forall t\,[\ {\sim}\text{STP}(f,x,t)\ ]$ NRNC
- $f \in \text{RangeAll} \Leftrightarrow \forall x \exists <y,t>\,[\ \text{STP}(f,y,t)\ \&\text{VALUE}(f,y,t)=x\ ]$ NRNC
- $f \in \text{RangeNotAll} \Leftrightarrow \exists x \forall <y,t>\,[\text{STP}(f,y,t) \Rightarrow \text{VALUE}(f,y,t)\neq x\ ]$ NRNC
- $f \in \text{HasZero} \Leftrightarrow \exists <x,t>\,[\ \text{STP}(f,x,t)\ \&\ \text{VALUE}(F,x,t)=0\ ]$ RE
- $f \in \text{IsZero} \Leftrightarrow \forall x \exists t\,[\ \text{STP}(f,x,t)\ \&\ \text{VALUE}(F,x,t)=0\ ]$ NRNC
- $f \in \text{Empty} \Leftrightarrow \forall <x,t>\,[\ {\sim}\text{STP}(f,x,t)\ ]$ Co-RE
- $f \in \text{NotEmpty} \Leftrightarrow \exists\ <x,t>\,[\ \text{STP}(f,x,t)\ ]$ RE

More Quantification Examples

- $f \in$ Identity $\Leftrightarrow \forall x \exists t$ [STP(f,x,t) & VALUE(f,x,t)=x] NRNC
- $f \in$ NotIdentity $\Leftrightarrow \exists x \forall t$ [~STP(f,x,t) | VALUE(f,x,t)≠x] or $\exists x \forall t$ [STP(f,x,t) $\Rightarrow$ VALUE(f,x,t)≠x] NRNC
- $f \in$ Constant = $\forall$ <x,y> $\exists t$ [STP(f,x,t) & STP(f,y,t) & VALUE(f,x,t)=VALUE(f,y,t)] NRNC
- $f \in$ Infinite $\Leftrightarrow \forall x \exists$ <y,t> [y≥x & STP(f,x,t)] NRNC
- $f \in$ Finite $\Leftrightarrow \exists x \forall$ <y,t> [y<x | ~STP(f,y,t)] or $\exists x \forall$ <y,t> [STP(f,y,t) $\Rightarrow$ y<x] or [y≥x $\Rightarrow$ ~STP(f,y,t)] NRNC
- $f \in$ RangeInfinite $\Leftrightarrow \forall x \exists$ <y,t> [STP(f,y,t) & VALUE(f,y,t)≥x] NRNC
- $f \in$ RangeFinite $\Leftrightarrow \exists x \forall$ <y,t> [STP(f,y,t) $\Rightarrow$ VALUE(f,y,t)<x] NRNC
- $f \in$ Stutter $\Leftrightarrow \exists$ <x,y,t> [x≠y & STP(f,x,t) & STP(f,y,t) & VALUE(f,x,t) = VALUE(f,y,t)] RE

Even More Quantification Examples

- **<f,x> $\in$ Fast20 $\Leftrightarrow$ [STP(f,x,20)]** **REC**
- **f $\in$ FastOne20 $\Leftrightarrow$ $\exists$x [STP(f,x,20)]** **RE**
- **f $\in$ FastAll20 $\Leftrightarrow$ $\forall$x [STP(f,x,20)]** **Co-RE**
- **<f,x,K,C> $\in$ LinearKC $\Leftrightarrow$ [STP(f,x,K*x+C)]** **REC**
- **<f,K,C>$\in$ LinearKCOne $\Leftrightarrow$ $\exists$x [STP(f,x,K*x+C)]** **RE**
- **<f,K,C> $\in$ LinearKCAll $\Leftrightarrow$ $\forall$x [STP(f,x,K*x+C)]** **Co-RE**

- **None of the above can be shown undecidable using Rice's Theorem**
- **In fact, reduction from known undecidables is also a problem.**

Some Reductions and Rice Example

- **NotEmpty ≤ Halt**
  Let f be an arbitrary index
  Define $\forall y\ g_f(y) = \exists$ <x,t> STP(f,x,t)
  $f \in$ ENotmpty $\Leftrightarrow$ <g_f,0> $\in$ Halt
- **Halt ≤ NotEmpty**
  Let f,x be an arbitrary index and input value
  Define $\forall y\ g_{f,x}(y) = f(x)$
  <f,x> $\in$ Halt $\Leftrightarrow g_{f,x} \in$ Empty
- **Note: NotEmpty is RE-Complete**
- **Rice: NotEmpty is non-trivial Zero $\in$ NotEmpty; $\uparrow \notin$ NotEmpty**
  Let f,g be arbitrary indices such that Dom(f)=Dom(g)
  $f \in$ NotEmpty $\Leftrightarrow$ Dom(f) $\neq \varnothing$ By Definition
  $\Leftrightarrow$ Dom(g) $\neq \varnothing$ Dom(g)=Dom(f)
  $\Leftrightarrow g \in$ NotEmpty
  Thus, Rice's Theorem states that NotEmpty is undecidable.

More Reductions and Rice Example

- **Identity ≤ Total**
  Let f be an arbitrary index
  Define $g_f(x) = \mu y\ [\ f(x) = x\ ]$
  $f \in$ Identity $\Leftrightarrow g_f \in$ Total
- **Total ≤ Identity**
  Let f be an arbitrary index
  Define $g_f(x) = f(x)-f(x) + x$
  $f \in$ Total $\Leftrightarrow g_{f,x} \in$ Identity
- **Rice: Identity is non-trivial $I(x)=x \in$ Identity; Zero$\notin$Identity**
  Let f,g be arbitrary indices such that $\forall x\ f(x) = g(x)$
  $f \in$ Identity $\Leftrightarrow \forall x\ f(x)=x$ By Definition
  $\Leftrightarrow \forall x\ g(x)=x$ $\forall x\ g(x) = f(x)$
  $\Leftrightarrow g \in$ Identity
  Thus, Rice's Theorem states that Identity is undecidable

Even More Reductions and Rice Example

- **Stutter ≤ Halt**
  Let f be an arbitrary index
  Define $\forall y\ g_f(y) = \exists \langle x,y,t\rangle$ [$x \neq y$ & STP(f,x,t) & STP(f,y,t) &
  VALUE(f,x,t) = VALUE(f,y,t)]
  $f \in$ Stutter $\Leftrightarrow \langle g_f,0\rangle \in$ Halt
- **Halt ≤ Stutter**
  Let f,x be an arbitrary index and input value
  Define $\forall y\ g_{f,x}(y) = f(x)$
  $\langle f,x\rangle \in$ Halt $\Leftrightarrow g_{f,x} \in$ Stutter
- **Note: Stutter is RE-Complete**
- **Rice: Stutter is non-trivial Zero $\in$ Stutter; $I(x)=x \notin$ Stutter**
  Let f,g be arbitrary indices such that $\forall x\ f(x) = g(x)$
  $f \in$ Stutter $\Leftrightarrow \exists \langle x,y\rangle$ [$x \neq y$ & $f(x)=f(y)$]
  $\Leftrightarrow \exists \langle x,y\rangle$ [$x \neq y$ & $g(x)=g(y)$] **By Definition $\forall x\ g(x) = f(x)$**
  $\Leftrightarrow g \in$ Stutter
  Thus, Rice's Theorem states that Identity is undecidable

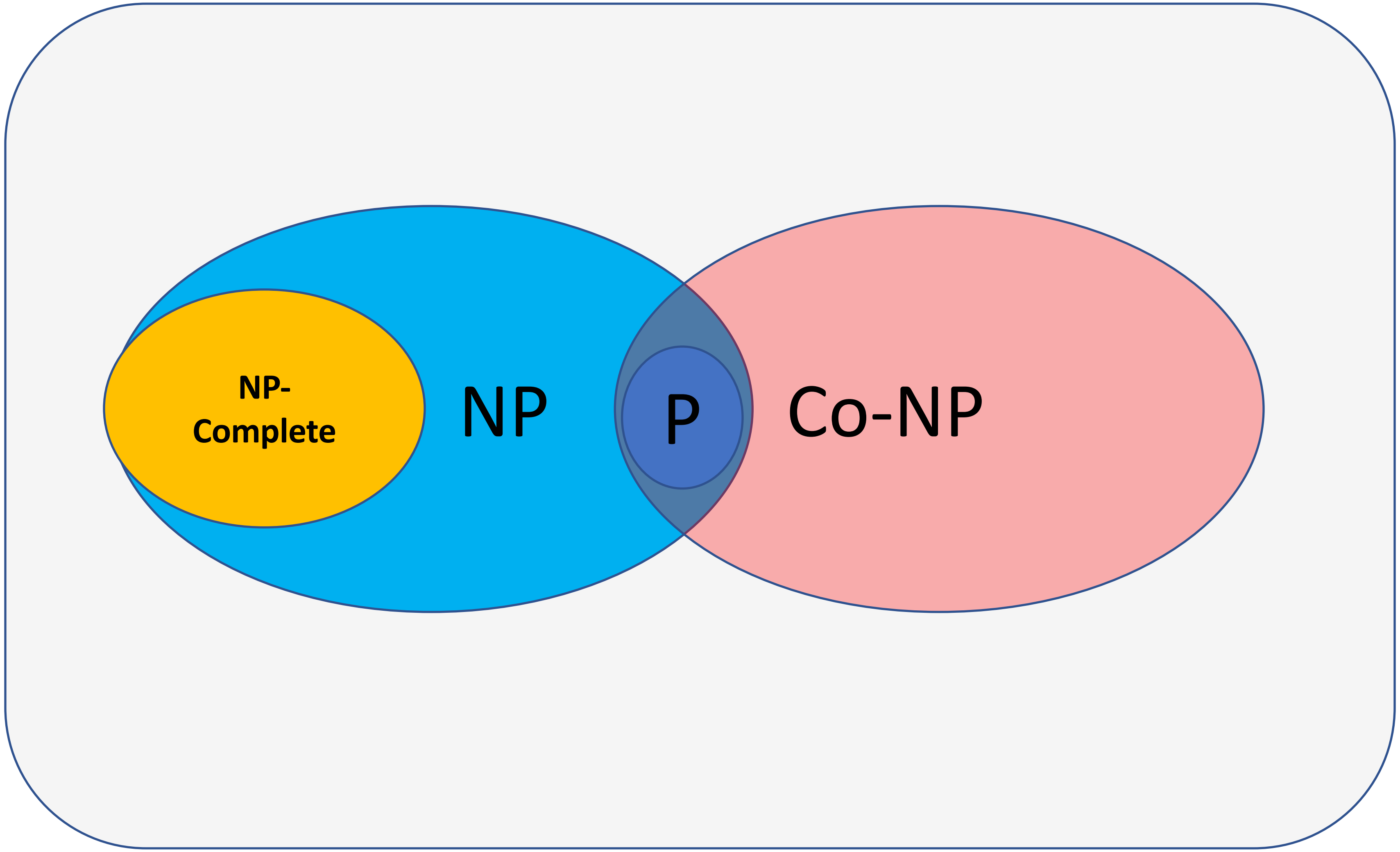
Yet More Reductions and Rice Example

- **Constant ≤ Total**
  Let f be an arbitrary index
  Define $g_f(0) = f(0)$
  $g_f(y+1) = \mu y\ [\ f(y+1) = f(y)\ ]$
  $f \in$ Constant $\Leftrightarrow g_f \in$ Total
- **Total ≤ Identity**
  Let f be an arbitrary index
  Define $g_f(x) = f(x)-f(x)$
  $f \in$ Total $\Leftrightarrow g_f \in$ Constant
- **Rice: Constant is non-trivial Zero $\in$ Constant; $I(x)=x \notin$ Constant**
  Let f,g be arbitrary indices such that $\forall x\ f(x) = g(x)$
  $f \in$ Constant $\Leftrightarrow \exists C \forall x\ f(x)=C$ By Definition
  $\Leftrightarrow \exists C \forall x\ g(x)=C$ $\forall x\ g(x) = f(x)$
  $\Leftrightarrow g \in$ Constant
  Thus, Rice's Theorem states that Identity is undecidable

Last Reductions and Rice Example

- **RangeAll ≤ Total**
  Let f be an arbitrary index
  Define $g_f(x) = \exists y\ [\ f(y) = x\ ]$
  $f \in$ RangeAll $\Leftrightarrow g_f \in$ Total
- **Total ≤ RangeAll**
  Let f be an arbitrary index
  Define $g_f(x) = f(x)-f(x) + x$
  $f \in$ Total $\Leftrightarrow g_f \in$ RangeAll
- **Rice: RangeAll is non-trivial $I(x)=x \in$ RangeAll; Zero $\notin$ RangeAll**
  Let f,g be arbitrary indices such that Range(f) = Range(g)
  $f \in$ RangeAll $\Leftrightarrow$ Range(f) = $\aleph$ By Definition
  $\Leftrightarrow$ Range(f) = $\aleph$ Range(g) = Range(f)
  $\Leftrightarrow g \in$ RangeAll
  Thus, Rice's Theorem states that Identity is undecidable

UNIVERSE OF SETS



Complexity Sample#1

#	Concept	Description	Concept #
1	Problem A is in NP	The classic NP-Complete problem	10
2	Problem A is in co-NP	A is the problem TOTAL (set of Algorithms)	4
3	Problem A is in P	A is decidable in deterministic polynomial time	3
4	Problem A is non-RE/non-Co-RE	If B is in NP then B $\le_P$ A	9
5	Problem A is NP-Complete	A is in RE and, if B is in RE, then B $\le_m$ A	8
6	Problem A is RE	A is verifiable in deterministic polynomial time	1
7	Problem A is Co-RE	A is in NP and if B is in NP then B $\le_P$ A	5
8	Problem A is RE-Complete	A is semi-decidable	6
9	Problem A is NP-Hard	A is the complement of B and B is RE	7
10	Satisfiability	A's complement is in NP	2

Sample#2: 3SAT to SubsetSum

(~a + b + ~c) (~a + ~b + c)

	a	b	c	~a + b + ~c	~a + ~b + c
a	1	0	0	0	0
~a	1	0	0	1	1
b	0	1	0	1	0
~b	0	1	0	0	1
c	0	0	1	0	1
~c	0	0	1	1	0
C1	0	0	0	1	0
C1’	0	0	0	1	0
C2	0	0	0	0	1
C2’	0	0	0	0	1
	1	1	1	3	3

Sample#3: Scheduling

List Schedule (T1,4), (T2,5), (T3,2), (T4,7), (T5,1), (T6,4), (T7,8)

T1	**T1**	**T1**	**T1**	**T3**	**T3**	**T5**	**T6**	**T6**	**T6**	**T6**	**T7**	**T7**	**T7**	**T7**	**T7**	**T7**	**T7**	**T7**
T2	T2	T2	T2	T2	T4	T4	T4	T4	T4	T4	T4							

Sorted List Schedule (T7,8), (T4,7), (T2,5), (T1,4), (T6,4), (T3,2), (T5,1)

T7	**T7**	**T7**	**T7**	**T7**	**T7**	**T7**	**T7**	**T1**	**T1**	**T1**	**T1**	**T6**	**T6**	**T6**	**T6**			
T4	T4	T4	T4	T4	T4	T4	T2	T2	T2	T2	T2	T3	T3	T5				

Independent set (IS) is NP-Complete

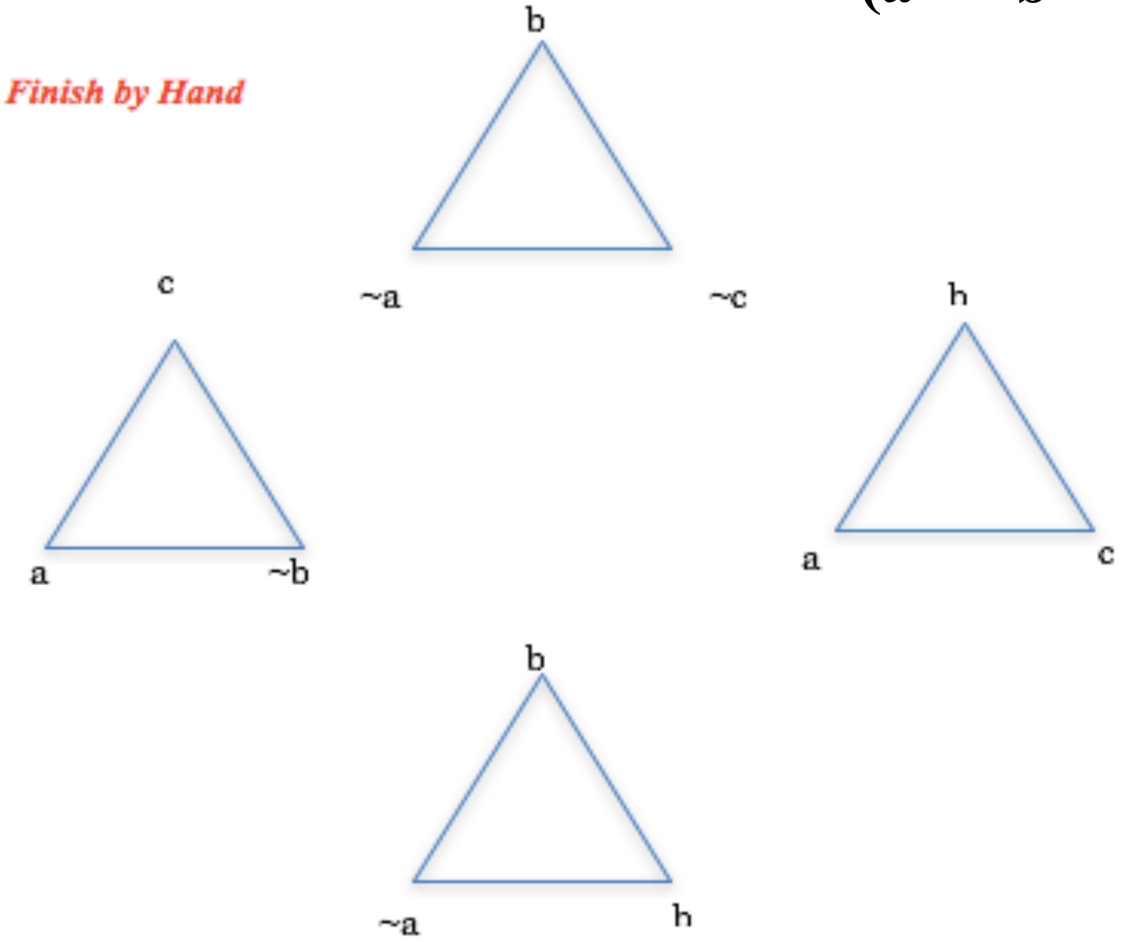
- We represent each clause in an instance of 3SAT with a triangle, one node per literal. The key is that all nodes are connected in a triangle of nodes, so the best you can do is to choose one node per clause to participate in an independent set. By adding an edge between every instance of variable v and every instance of variable ~v, we guarantee that we cannot choose nodes labeled v and ~v as part of an independent set. Here, assume we have V Boolean variables
- When the required independent set must be C, where C is the number of clauses, we must choose one node per clause and we must do this in a way so that no nodes labeled with a variable and its complement are chosen. That can only be done if there is an assignment to variables (true or false) that satisfy the original instance of 3SAT. Thus IS is NP-Hard. But, we can check a proposed independent set in time proportional to the size of the graph (which is actually linear in the size of the 3SAT problem). Thus IS is in P. In conclusion, IS is NP-Complete.

Sample#4: Independent Set

k=4

Finish by Hand

$$(a + \sim b + c)\ (\sim a + b + \sim c)\ (a + b + c)\ (\sim a + b + b)$$

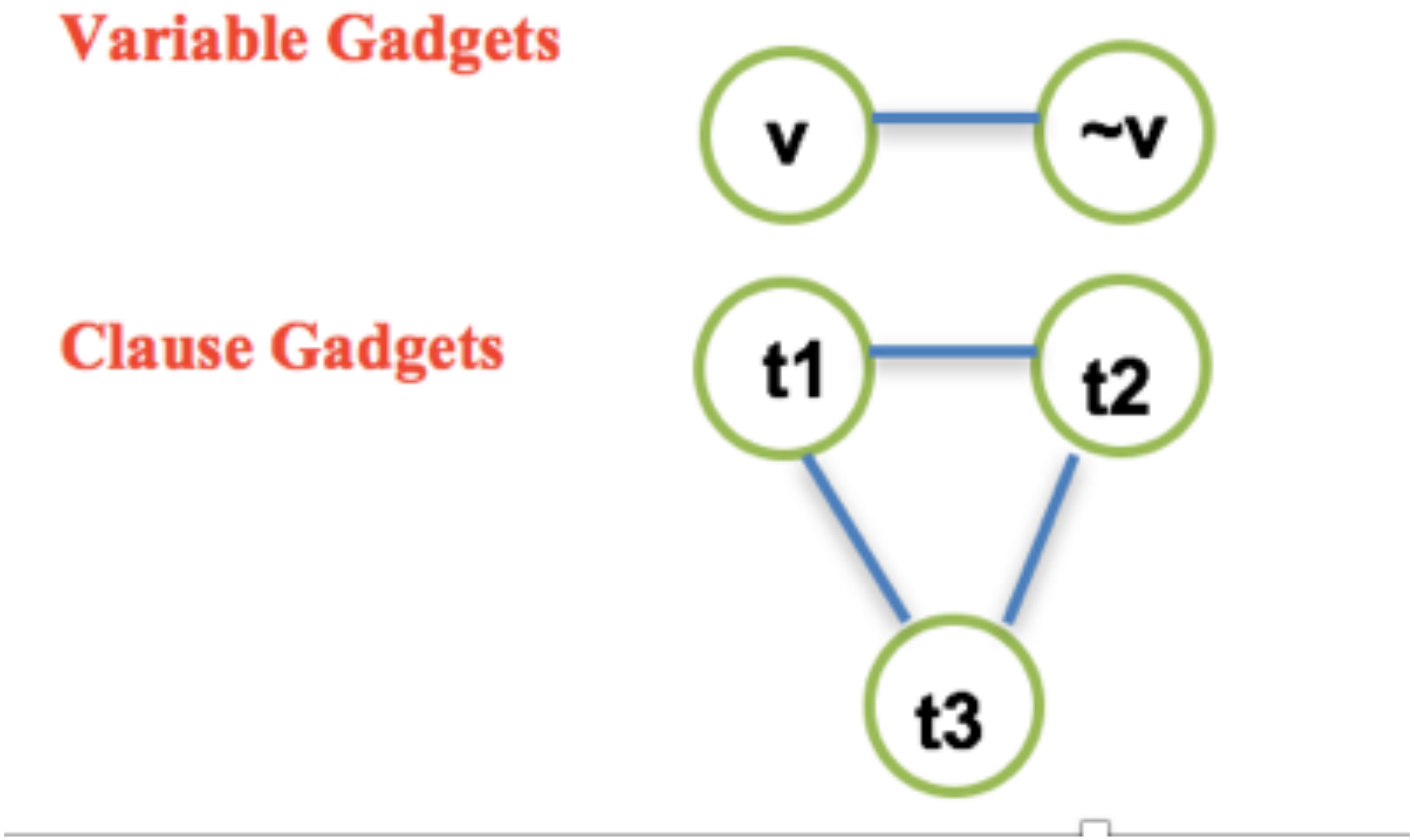


Place an edge between every node labeled V and every node labeled ~V, where V can be a, b or c.

Vertex Cover (VC) is NP-Complete

- We represent each clause (assume there are C of them) in an instance of 3SAT with a triangle, one node per literal. One key is that two nodes in each clause triangle must be chosen to cover the three internal edges. We represent each assignment to a variable v (assume there are V variables) by a pair of connected nodes labeled v and ~v. The second key is that we must choose precisely one of v or ~v for each variable to cover the edge that connects its pair. Thus, the minimum cover set contains 2C+V nodes.
- We add an edge from each v and to all literals v in clauses, and each ~v to all literals ~v in clauses. To cover all the edges added here for the variable nodes, we must choose nodes in each clause that cover edges from variable nodes that are not chosen in the variable pair. If all clauses have at least one of these incoming edges already covered (we chose an assignment to the variable that matches a literal in this clause), then we will be able to cover all internal edges in each clause and all edges entering the clause from a variable pair, by just choosing two nodes in the clause.
- Choosing 2C+V nodes that cover all edges can only be done if there is an assignment to variables (true or false) that satisfy the original instance of 3SAT. Thus VC is NP-Hard. But, we can check a proposed cover set of vertices in time proportional to the size of the graph (which is actually linear in the size of the 3SAT problem). Thus VC is in P. In conclusion, VC is NP-Complete.

Sample # 5: VC Gadgets

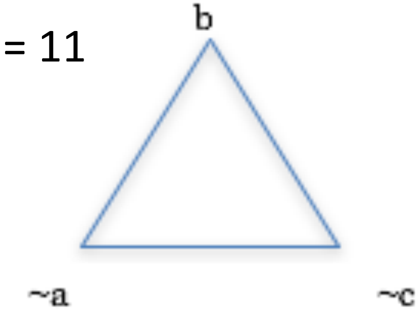


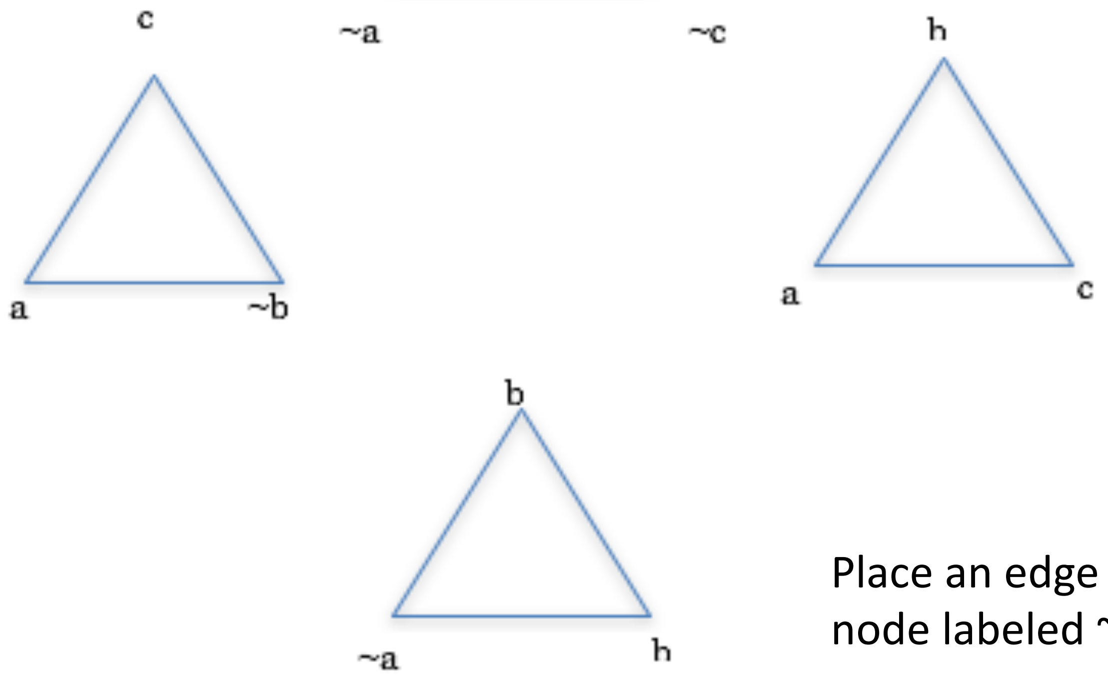
Sample#6: Vertex Cover

K = 2*C+V = 8+3 = 11

Finish by Hand

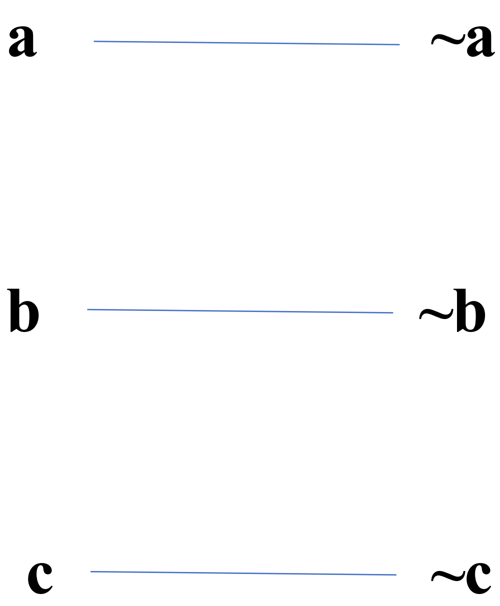
Clause Nodes/Edges





(a + ~b + c) (~a + b + ~c) (a + b + c) (~a + b + b)

Variable Nodes/Edges



Place an edge between every variable node labeled V and every clause node labeled ~V, where V can be a, b or c.

The meaning of “defined” in Halt discussion

- In the diagonalization proof that the Halting Problem is undecidable, you use the term defined, as in Disagree(d) is defined iff Halt(d,d)=0 iff Disagree(d) is undefined. What is its meaning in this context?

  The word “defined” here means "converges and defines a value." The procedure "least value of y such that y=y+1" can never converge and so never defines a value.
- Fortunately, I will not ask you to repeat this proof, but you need to understand its significance. That is, you need to remember where it was used and for what purposes.

RE and Co-RE

- Why can't the complement of something that is re non-recursive be recursive?
- The reason is that any set that is RE and whose complement is RE, is also a recursive set. Thus, if S is RE, non-recursive, its complement must be co-RE, non-recursive.

Characteristic Function for a Decidable Set

- By definition of decidable, S is decidable iff there's a characteristic function. Could you explain what is a characteristic function, with examples?

  The term "characteristic function" for some recursive/decidable set, S, just refers to any algorithmic predicate to decide membership in S. The term is used as the algorithm "charcaterizes" S by allowing us to decided its membership.

Union of sets

- Let set A be recursive, B be re non-recursive and C be non-re, what can D be if D is contained in (A U C).

  Consider the case where A is the set of natural numbers, then (A U C) is the set of all natural numbers, no matter what C is. If D is a subset of the set of natural numbers then, D can be anything. For instance, D can be empty, in which case it is recursive; D could be the set of indices of functions that halt on some input, in which case it is RE; D could be the set of indices of algorithms, in which case it it is non-RE. In fact, there are an uncountably infinite number of subsets of the natural numbers, so there is no telling what D might be.

Phrase Structured Grammars

- Are we doing anything on phrase structured grammars?

  There will be no PSG's to write. I never had time to do anything on that. The only thing that is critical is to remember that the Phrase Structured Languages are exactly the RE sets.

Containment

- You say in the exam review that containiment is undecidable for CFL's. But, on the 2nd exam the answer to #3 says that L(G) contains and may equal {λ} and that this is a decidable problem. Why is this?

  On the review slides, the only place that we said containment is decidable is for Regular Languages. Page 3 explicitly states that containment id undecidable for CFLs. Exam#2 has no mention of containment.

  What is .decidable is membership. Membership is testing a single or finite set of strings for membership in the language -- decidable. Containment refers to whether or not a language is a subset of another. That is undeciadble for CFLs. The proof is based on the fact that we cannot decide of an arbitrary CFG if it generates Sigma*. But then we cannot decide if Sigma* is a subset of some arbitrary context-free language.

Edges added to Graphs in IS and VC

- On the vertex covering problem it says to "place an edge between every variable node labeled V and every clause node labeled ~V". During your office hours, you showed on example to us of VC but you placed an edge between every node labeled V and every clause node labeled V as well. Is there no difference in the way the connections between clauses are made between the triangles in Independent Set problems and VC problems?

  IS is all v to ~v. VC is v to ~v in the variable pair gadgets and v to v, ~v to ~v in the edges between the variable pair gadgets and the clause gadgets. As I read my review, this is exactly what it says.

Pumping Lemma

- I noticed Pumping Lemma isn't on the review. Do we need to know it?

  You need to understand the concepts of the two Pumping Lemmas but I will not ask an explicit question to apply either. By concepts I mean, the essence of their proofs and how they are applied.