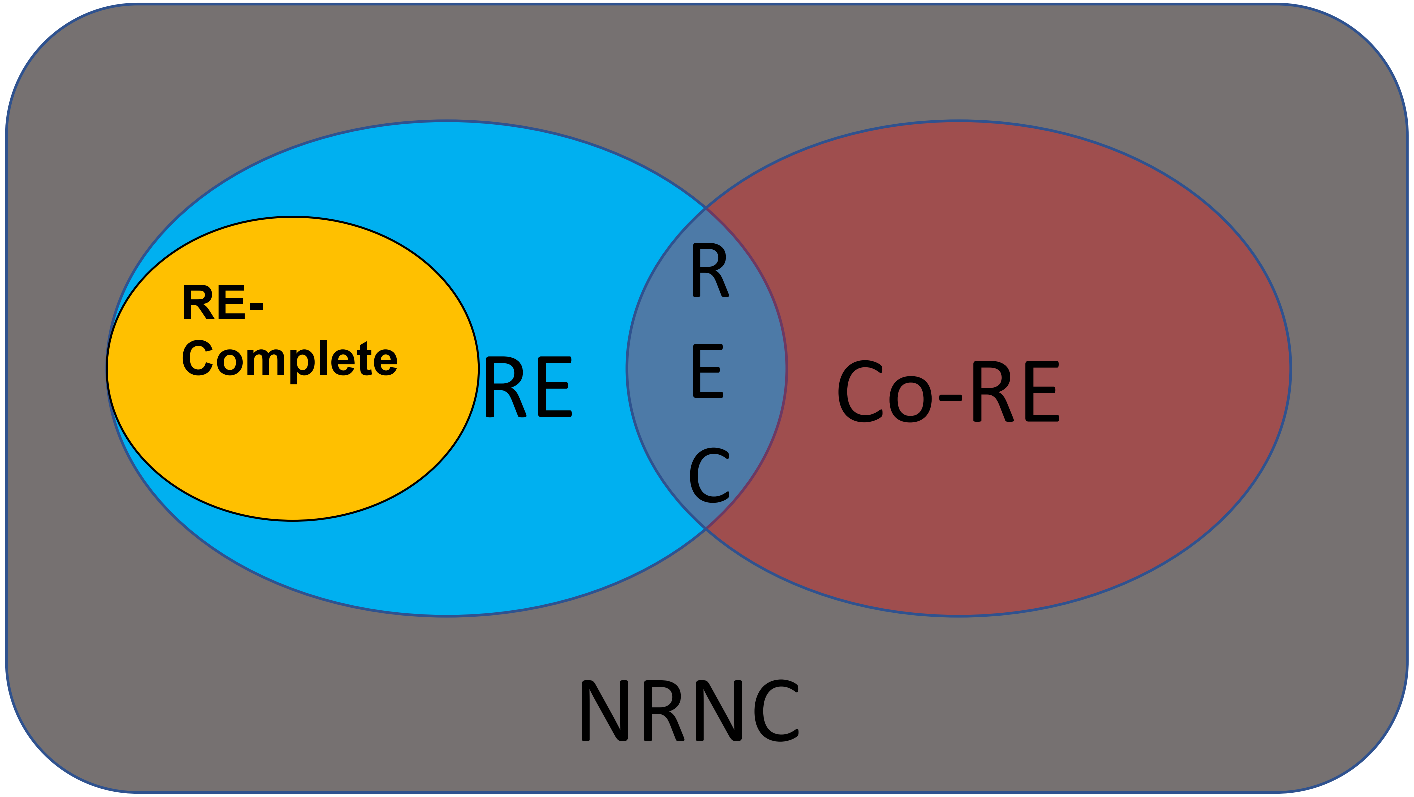
UNIVERSE OF SETS



NR (non-recursive)

= (NRNC U Co-RE) - REC

Some Quantification Examples

• <f,x> ∈ Halt ⇔ ∃t [STP(f,x,t)]	RE
• f ∈ Total ⇔ ∀x∃t [STP(f,x,t)]	NRNC
• f ∈ NotTotal ⇔ ∃x∀t [~STP(f,x,t)]	NRNC
• f ∈ RangeAll ⇔ ∀x∃<y,t> [STP(f,y,t) &VALUE(f,y,t)=x]	NRNC
• f ∈ RangeNotAll ⇔ ∃x∀<y,t> [STP(f,y,t) ⇒ VALUE(f,y,t)≠x]	NRNC
• f ∈ HasZero ⇔ ∃<x,t> [STP(f,x,t) & VALUE(f,x,t)=0]	RE
• f ∈ IsZero ⇔ ∀x∃t [STP(f,x,t) & VALUE(f,x,t)=0]	NRNC
• f ∈ Empty ⇔ ∀<x,t> [~STP(f,x,t)]	Co-RE
• f ∈ NotEmpty ⇔ ∃ <x,t> [STP(f,x,t)]	RE

More Quantification Examples

- $f \in \text{Identity} \Leftrightarrow \forall x \exists t\ [\ STP(f,x,t)\ \&\ VALUE(f,x,t)=x\ ]$ NRNC
- $f \in \text{NotIdentity} \Leftrightarrow \exists x \forall t\ [\ \sim STP(f,x,t)\ |\ VALUE(f,x,t) \neq x\ ]$ or $\exists x \forall t\ [\ STP(f,x,t) \Rightarrow VALUE(f,x,t) \neq x\ ]$ NRNC
- $f \in \text{Constant} = \forall \langle x,y \rangle \exists t\ [STP(f,x,t)\ \&\ STP(f,y,t)\ \&\ VALUE(f,x,t)=VALUE(f,y,t)]$ NRNC
- $f \in \text{Infinite} \Leftrightarrow \forall x \exists \langle y,t \rangle\ [\ y \geq x\ \&\ STP(f,y,t)\ ]$ NRNC
- $f \in \text{Finite} \Leftrightarrow \exists x \forall \langle y,t \rangle\ [\ y<x\ |\ \sim STP(f,y,t)\ ]$ or $\exists x \forall \langle y,t \rangle\ [\ STP(f,y,t) \Rightarrow y<x\ ]$ or $[\ y \geq x \Rightarrow\ \sim STP(f,y,t)\ ]$ NRNC
- $f \in \text{RangeInfinite} \Leftrightarrow \forall x \exists \langle y,t \rangle\ [\ STP(f,y,t)\ \&\ VALUE(f,y,t) \geq x\ ]$ NRNC
- $f \in \text{RangeFinite} \Leftrightarrow \exists x \forall \langle y,t \rangle\ [\ STP(f,y,t) \Rightarrow VALUE(f,y,t)<x\ ]$ NRNC
- $f \in \text{Stutter} \Leftrightarrow \exists \langle x,y,t \rangle\ [\ x \neq y\ \&\ STP(f,x,t)\ \&\ STP(f,y,t)\ \&\ VALUE(f,x,t) = VALUE(f,y,t)\ ]$ RE

Even More Quantification Examples

• <f,x> ∈ Fast20 ⇔ [STP(f,x,20)]	REC
• f ∈ FastOne20 ⇔ ∃x [STP(f,x,20)]	RE
• f ∈ FastAll20 ⇔ ∀x [STP(f,x,20)]	Co-RE
• <f,x,K,C> ∈ LinearKC ⇔ [STP(f,x,K*x+C)]	REC
• <f,K,C>∈ LinearKCOne ⇔ ∃x [STP(f,x,K*x+C)]	RE
• <f,K,C> ∈ LinearKCAll ⇔ ∀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 for all but the first one which happens to be decidable.

Some Reductions and Rice Example

- **NotEmpty ≤ Halt**
  Let f be an arbitrary index
  Define $\forall y\ g_f(y) = \exists\langle x,t\rangle\ STP(f,x,t)$
  $f \in$ Notmpty $\Leftrightarrow \langle g_f,0\rangle \in$ Halt
- **Halt ≤ NotEmpty**
  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 $\Leftrightarrow g_{f,x} \in$ NotEmpty
- **Note: NotEmpty is RE-Complete**
- **Rice: NotEmpty is non-trivial Zero ∈ NotEmpty; $\uparrow \notin$ NotEmpty**
  Let f,g be arbitrary indices such that Dom(f)=Dom(g)
  $f \in$NotEmpty $\Leftrightarrow$ Dom(f) $\neq \emptyset$ By Definition
  $\Leftrightarrow$ Dom(g) $\neq \emptyset$ 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∈Stutter; $I(x)=x \notin$ Stutter**
  Let f,g be arbitrary indices such that $\forall x\ f(x) = g(x)$
  f ∈Stutter ⇔ $\exists\langle x,y\rangle$ [$x \neq y$ & $f(x)=f(y)$] By Definition
  ⇔ $\exists\langle x,y\rangle$ [$x \neq y$ & $g(x)=g(y)$] $\forall x\ g(x) = f(x)$
  ⇔ g ∈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 $g_f(0) = f(0)$
  $g_f(y+1) = \mu z\ [\ f(y+1) = f(y)\ ]$
  $f \in \text{Constant} \Leftrightarrow g_f \in \text{Total}$
- **Total ≤ Identity**
  Let f be an arbitrary index
  Define $g_f(x) = f(x)-f(x)$
  $f \in \text{Total} \Leftrightarrow g_f \in \text{Constant}$
- **Rice: Constant is non-trivial Zero ∈ Constant; $I(x)=x \notin$ Constant**
  Let f,g be arbitrary indices such that $\forall x\ f(x) = g(x)$
  $f \in \text{Constant} \Leftrightarrow \quad \exists C \forall x\ f(x)=C \quad$ By Definition
  $\Leftrightarrow \quad \exists C \forall x\ g(x)=C \quad \forall x\ g(x) = f(x)$
  $\Leftrightarrow g \in \text{Constant}$
  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(g) = $\aleph$ Range(g) = Range(f)
  $\Leftrightarrow g \in$ RangeAll
  Thus, Rice's Theorem states that Identity is undecidable

Challenge

Semi-Constant(SC) = { f | ∃C, ∀x f(x)↓ ⇒ f(x) = C }

Note: **↑ ∈ SC** and **C_0(x)=0 ∈ SC**

Can describe as **f ∈ SC** ⇔

∃C ∀<x,t> [STP(f,x,t) ⇒ VALUE(f,x,t) = C]

This implies **SC** is as hard as **Non-TOT={ f |∃x f(x)↑ }** as

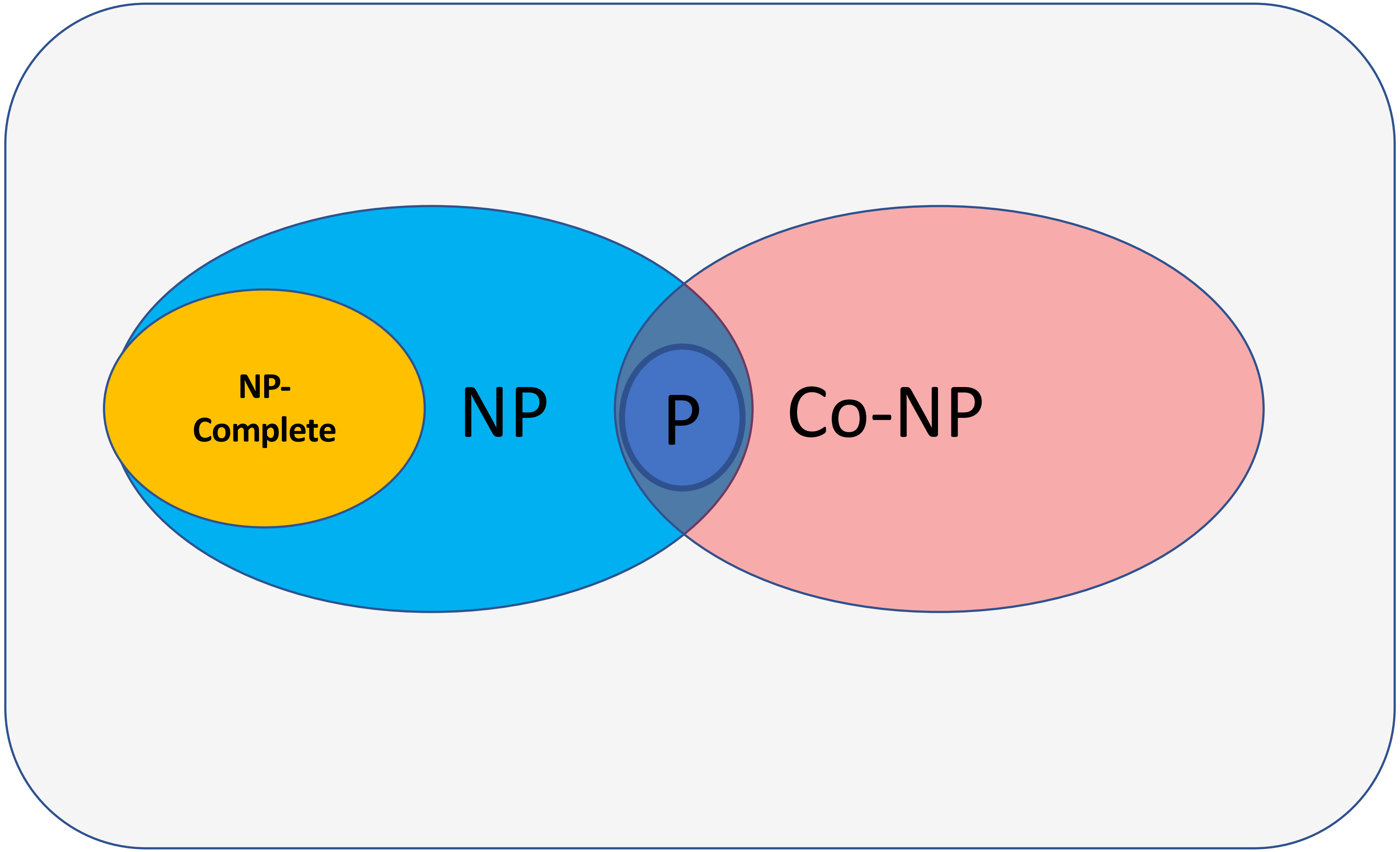
f ∈ Non-TOT ⇔ ∃x ∀t [~STP(f,x,t)]

However, **SC** only takes one quantifier and is undecidable (one of the weaker versions of Rice shows its undecidability).

I can tell you that **SC $\equiv_m$ HALT** or **SC $\equiv_m$ Non-HALT** where **Non-HALT = { <f,x> | f(x)↑ }**.

Your job is to figure out which and rewrite the quantifier expression. You should also apply Rice's to verify undecidability.

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 $B \leq_P A$	9
5	Problem A is NP-Complete	A is in RE and, if B is in RE, then $B \leq_m A$	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 $B \leq_P A$	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

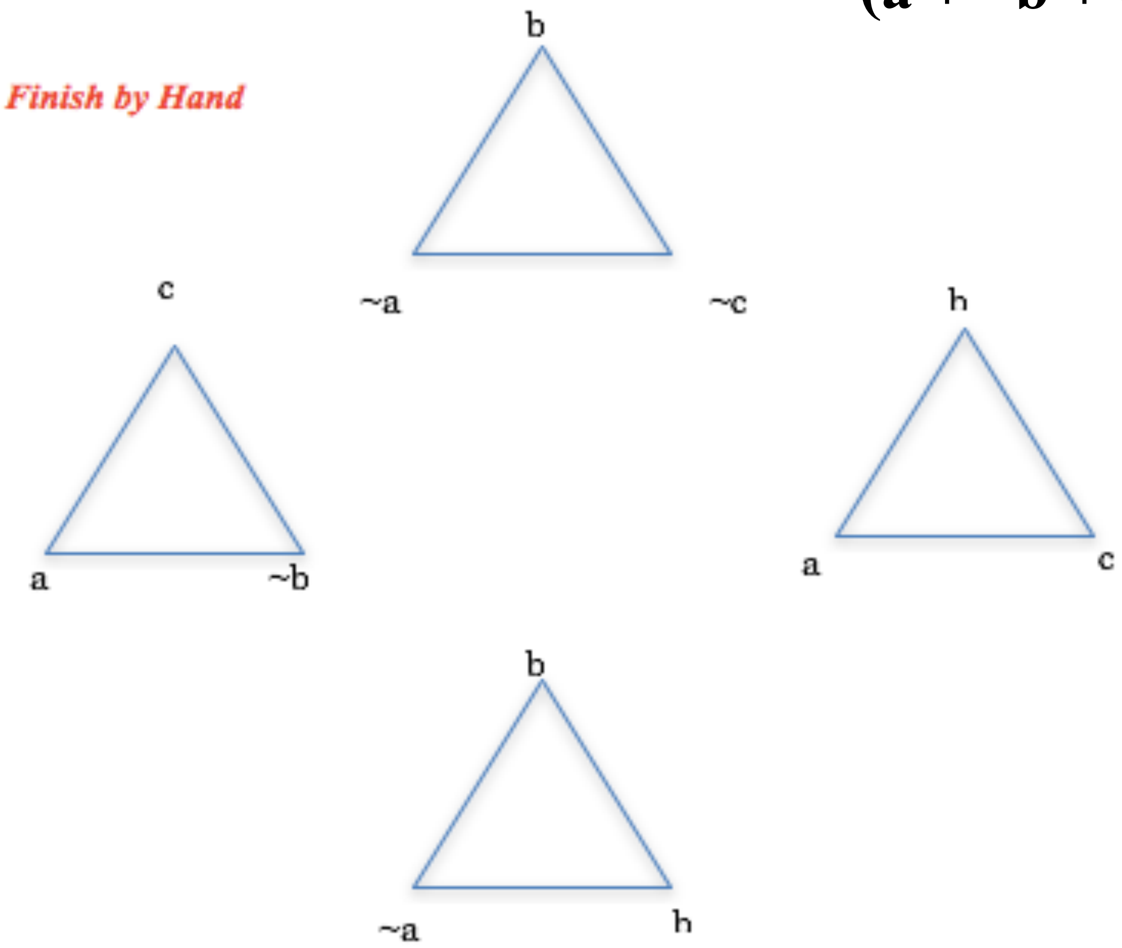
- We represent each clause in an instance of 3SAT with a triangle, one node per literal. The key is that all nodes are connected in a triangle of nodes, so the best you can do is to choose one node per clause to participate in an independent set. By adding an edge between every instance of variable v 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 NP. In conclusion, IS is NP-Complete.

Sample#4: Independent Set

k=4

Finish by Hand

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

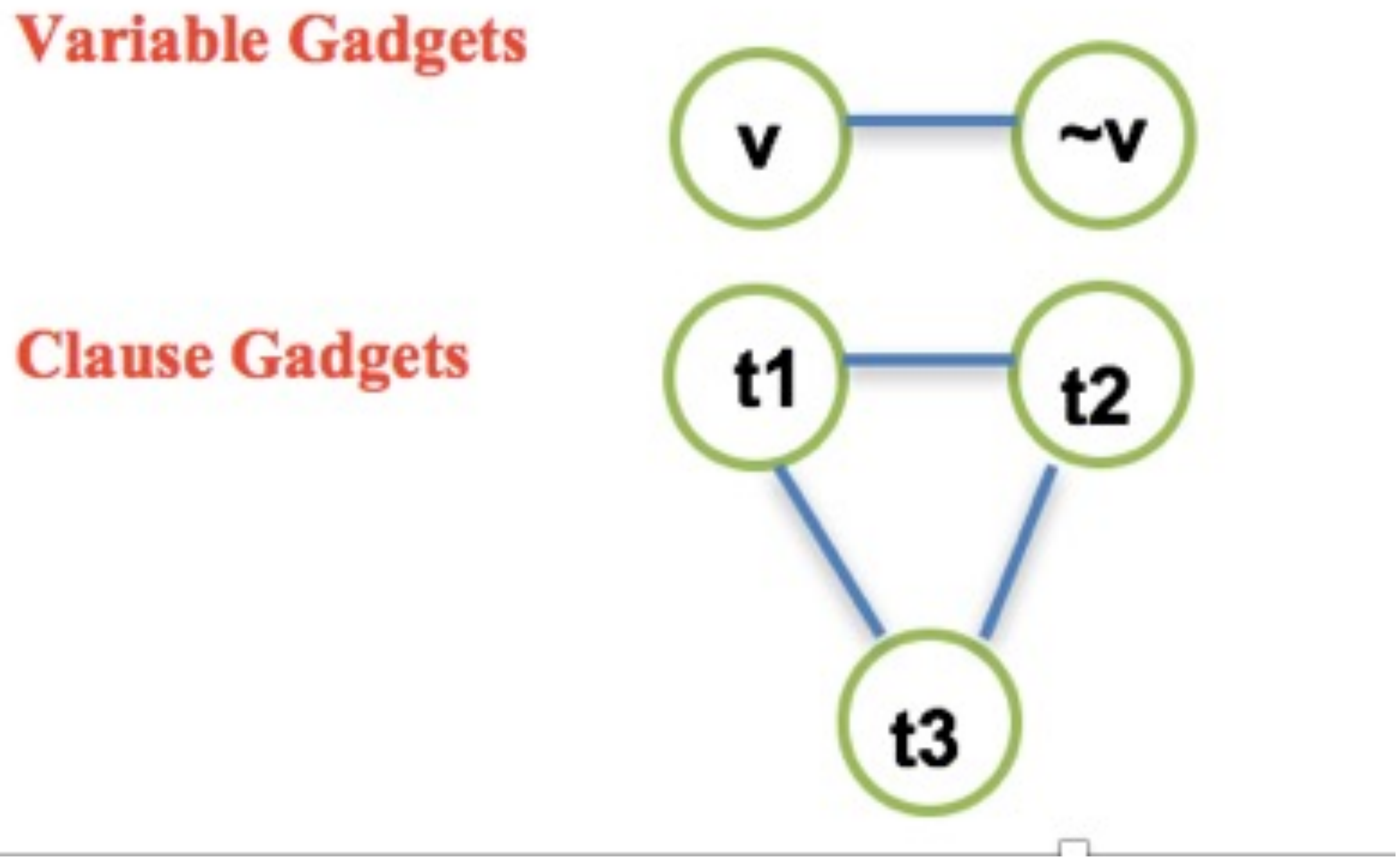


Place an edge between every node labeled V and every node labeled ~V, where V can be a, b or c.

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 NP. In conclusion, VC is NP-Complete.

Sample # 5: VC Gadgets

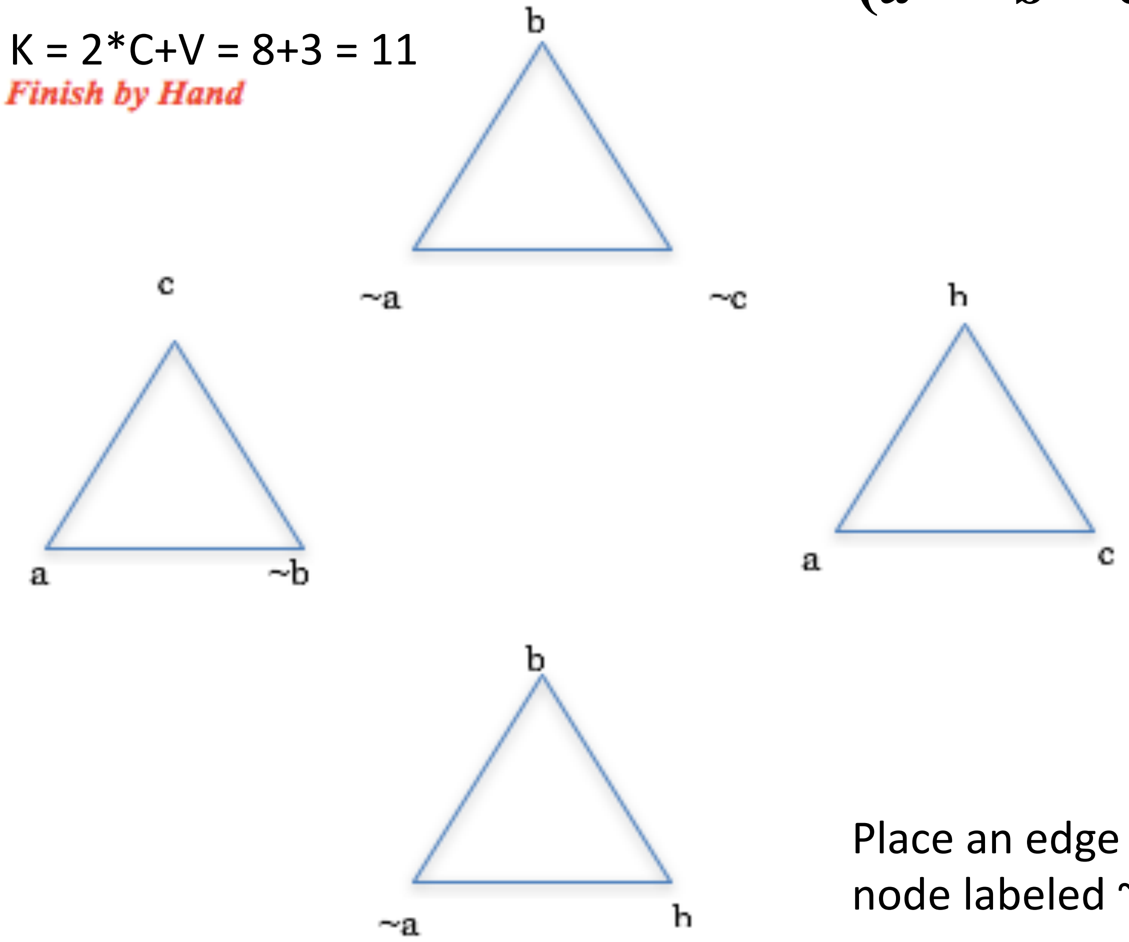


Sample#6: Vertex Cover

Clause Nodes/Edges

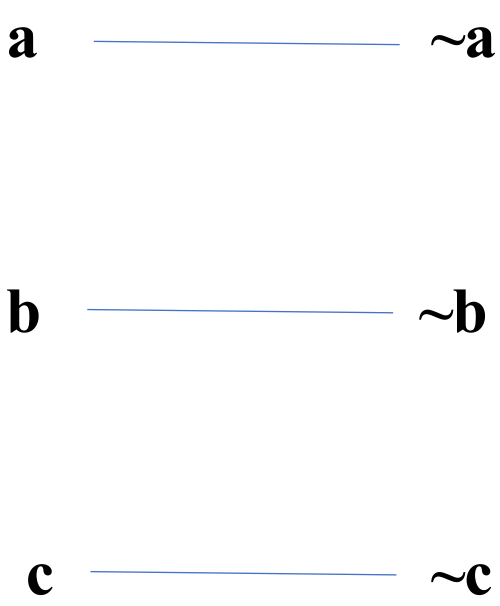
K = 2*C+V = 8+3 = 11

Finish by Hand



(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.

Consider the SAT instance:

(x1 ∨ x2 ∨ x4 ∨ x5) & (¬x1 ∨ ¬x2 ∨ x3 ∨ ¬x4 ∨ ¬x5) & (x1 ∨ ¬x4)

1. Recast this as an instance of 3SAT.

ANS:

(x1 ∨ x2 ∨ x6) & (x4 ∨ x5 ∨ ¬x6) & (¬x1 ∨ ¬x2 ∨ x7) & (x3 ∨ ¬x4 ∨ x8) & (¬x5 ∨ ¬x7 ∨ ¬x8) & (x1 ∨ ¬x4 ∨ x1)

ANS:

c1 = (x1 ∨ x2 ∨ x6)
c2 = (x4 ∨ x5 ∨ ¬x6)
c3 = (¬x1 ∨ ¬x2 ∨ x7)
c4 = (x3 ∨ ¬x4 ∨ x8)
c5 = (¬x5 ∨ ¬x7 ∨ ¬x8)
c6 = (x1 ∨ ¬x4 ∨ x1)

A simple solution is **x1, x2, x3, x4, x5, x6, x7, ¬x8**

2. Construct the SubsetSum instance equivalent to this and state what rows must be chosen.

(x1 V x2 V x6) & (x4 V x5 V ¬x6) & (¬x1 V ¬x2 V x7) & (x3 V ¬x4 V x8) & (¬x5 V ¬x7 V ¬x8) & (x1 V ¬x4 V x1)

	x1	x2	x3	x4	x5	x6	x7	x8	C1	C2	C3	C4	C5	C6
x1	1	0	0	0	0	0	0	0	1	0	0	0	0	2
~x1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
x2	0	1	0	0	0	0	0	0	1	0	0	0	0	0
~x2	0	1	0	0	0	0	0	0	0	0	1	0	0	0
x3	0	0	1	0	0	0	0	0	0	0	0	1	0	0
~x3	0	0	1	0	0	0	0	0	0	0	0	0	0	0
x4	0	0	0	1	0	0	0	0	0	1	0	0	0	0
~x4	0	0	0	1	0	0	0	0	0	0	0	1	0	1
x5	0	0	0	0	1	0	0	0	0	1	0	0	0	0
~x5	0	0	0	0	1	0	0	0	0	0	0	0	1	0
x6	0	0	0	0	0	1	0	0	1	0	0	0	0	0
~x6	0	0	0	0	0	1	0	0	0	1	0	0	0	0
x7	0	0	0	0	0	0	1	0	0	0	1	0	0	0
~x7	0	0	0	0	0	0	1	0	0	0	0	0	1	0
x8	0	0	0	0	0	0	0	1	0	0	0	1	0	0
~x8	0	0	0	0	0	0	0	1	0	0	0	0	1	0
C1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
C1’	0	0	0	0	0	0	0	0	1	0	0	0	0	0
C2	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C2’	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C3	0	0	0	0	0	0	0	0	0	0	1	0	0	0
C3 ‘	0	0	0	0	0	0	0	0	0	0	1	0	0	0
C4	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C4’	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C5	0	0	0	0	0	0	0	0	0	0	0	0	1	0
C5’	0	0	0	0	0	0	0	0	0	0	0	0	1	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C6’	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	1	1	1	1	1	1	1	1	3	3	3	3	3	3

3. Recast the SubsetSum instance in Part 2 as a Partition instance (really easy). Show the Partitioning into equal subsets.

Ans:
G = 11111111333333
sum= 22222222555555
$2 * sum - G =$ 33333333777777
sum + G = 33333333888888
sum is the sum of all rows.
Note: If you use 1 in X1/C6 then
sum is 22222222555554 and so
$2 * sum - G =$ 33333333777775
sum + G = 33333333888887
The partitions for the case where we use 2 in x1/C6 are as follows:

Partition 1:

33333333 777777	2*sum -G
10000000 100002	x1
01000000 100000	x2
00100000 000100	x3
00010000 010000	x4
00001000 010000	x5
00000100 100000	x6
00000010 001000	x7
00000001 000010	¬x8
00000000 010000	C2
00000000 010000	C3
00000000 001000	C3'
00000000 000100	C4
00000000 000010	C5
00000000 000010	C5'
00000000 000001	C6

c1 = (x1 V x2 V x6)
c2 = (x4 V x5 V ¬x6)
c3 = (¬x1 V ¬x2 V x7)
c4 = (x3 V ¬x4 V x8)
c5 = (¬x5 V ¬x7 V ¬x8)
c6 = (x1 V ¬x4 V x1)

A simple solution is **x1, x2, x3, x4, x5, x6, x7, ¬x8**

Partition 2:

33333333 888888	sum+G
10000000 001000	~x1
01000000 001000	~x2
00100000 000000	~x3
00010000 000101	~x4
00001000 000010	~x5
00000100 010000	~x6
00000010 000010	~x7
00000001 000100	x8
00000000 100000	C1
00000000 100000	C1’
00000000 010000	C2’
00000000 000100	C4’
00000000 000001	C6’

c1 = (x1 V x2 V x6)
c2 = (x4 V x5 V ¬x6)
c3 = (¬x1 V ¬x2 V x7)
c4 = (x3 V ¬x4 V x8)
c5 = (¬x5 V ¬x7 V ¬x8)
c6 = (x1 V ¬x4 V x1)

A simple solution is **x1, x2, x3, x4, x5, x6, x7, ¬x8**

4. Recast the original SAT as a 0-1 Integer Linear Programming instance:

(x1 ∨ x2 ∨ x4 ∨ x5) & (¬x1 ∨ ¬x2 ∨ x3 ∨ ¬x4 ∨ ¬x5) & (x1 ∨ ¬x4)

ANS:

Assume 0 <= x1, x2, x3, x4, x5 <= 1
x1 + x2 + x4 + x5 >= 1
(1-x1) + (1-x2) + x3 + (1-x4) + (1-x5) >= 1
x1 + (1-x4) >= 1
We choose: **x1 = 1, x2 = 1, x3 = 1, x4 = 1, x5 = 1**

5. Consider the following set of independent tasks with associated task times:

(T1,3), (T2,5), (T3,7), (T4,6), (T5,2), (T6,8), (T7,1)

Fill in the schedules for these tasks under the associated strategies below.

Greedy using the list order above:

Greedy using a reordering of the list so that longest-running tasks appear earliest in the list:

Greedy then sorted high to low

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

(T1,3), (T2,5), (T3,7), (T4,6), (T5,2), (T6,8), (T7,1)

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

(T6,8), (T3,7), (T4,6), (T2,5), (T1,3), (T5,2), (T7,1)

6. Consider the 3SAT instance:

E = ($x1 \lor x2 \lor x4$) & ($\neg x1 \lor \neg x3 \lor \neg x4$) & ($\neg x2 \lor \neg x3 \lor x4$) & ($\neg x2 \lor \neg x3 \lor \neg x4$)

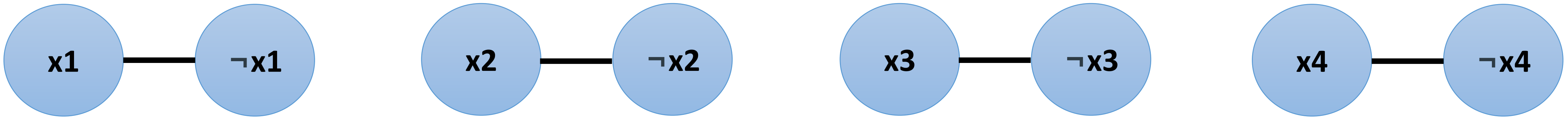
a. Recast **E** as an instance of k-Vertex Covering and present a solution to the latter

b. Recast **E** as an instance of 3-Coloring and present a solution to the latter

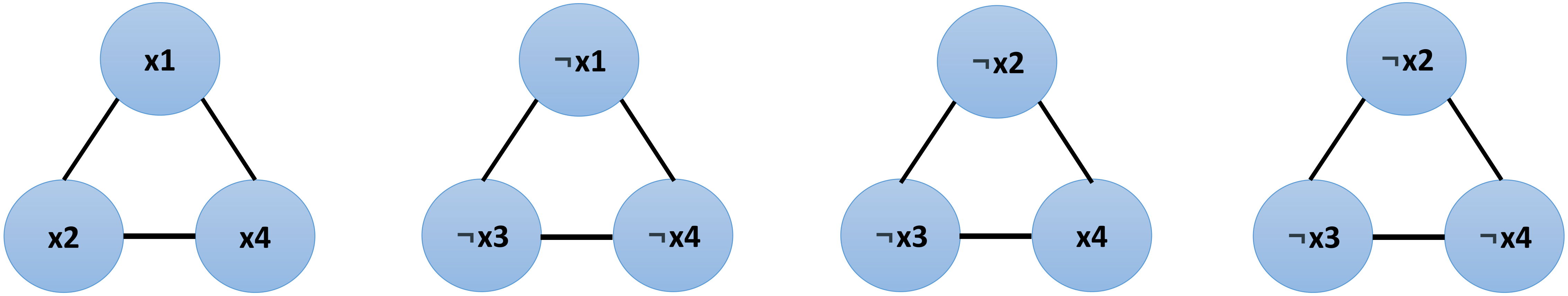
Question 6 (a)

E = (x1 ∨ x2 ∨ x4) & (¬x1 ∨ ¬x3 ∨ ¬x4) & (¬x2 ∨ ¬x3 ∨ x4) & (¬x2 ∨ ¬x3 ∨ ¬x4)

Variable Gadgets:

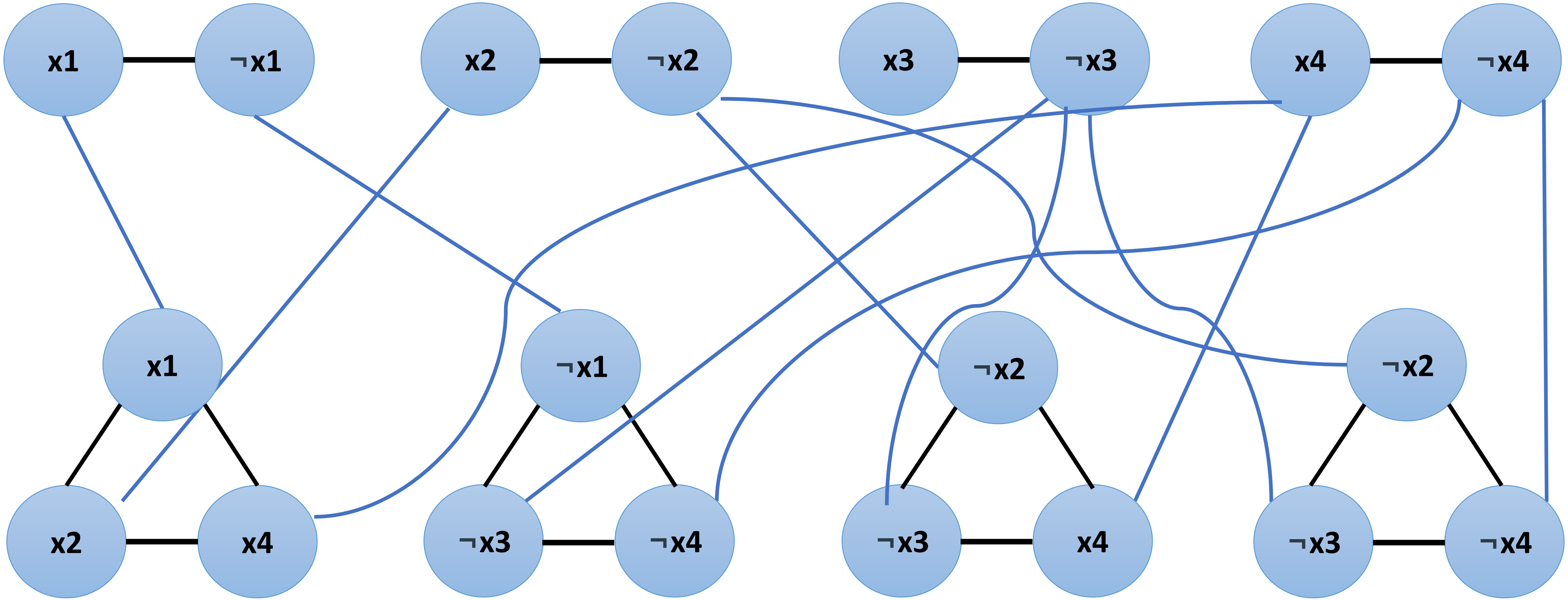


Clause Gadgets:



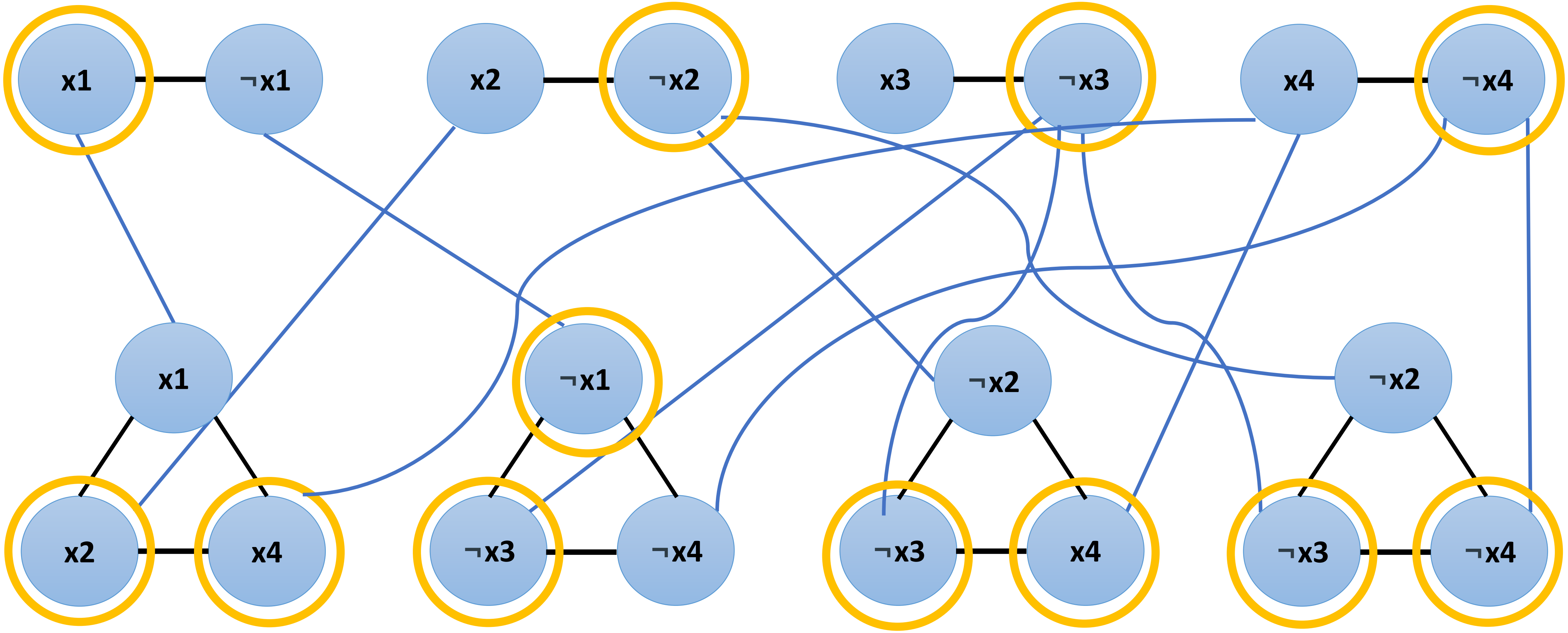
E = (x1 ∨ x2 ∨ x4) & (¬x1 ∨ ¬x3 ∨ ¬x4) & (¬x2 ∨ ¬x3 ∨ x4) & (¬x2 ∨ ¬x3 ∨ ¬x4)

Combined Gadgets:

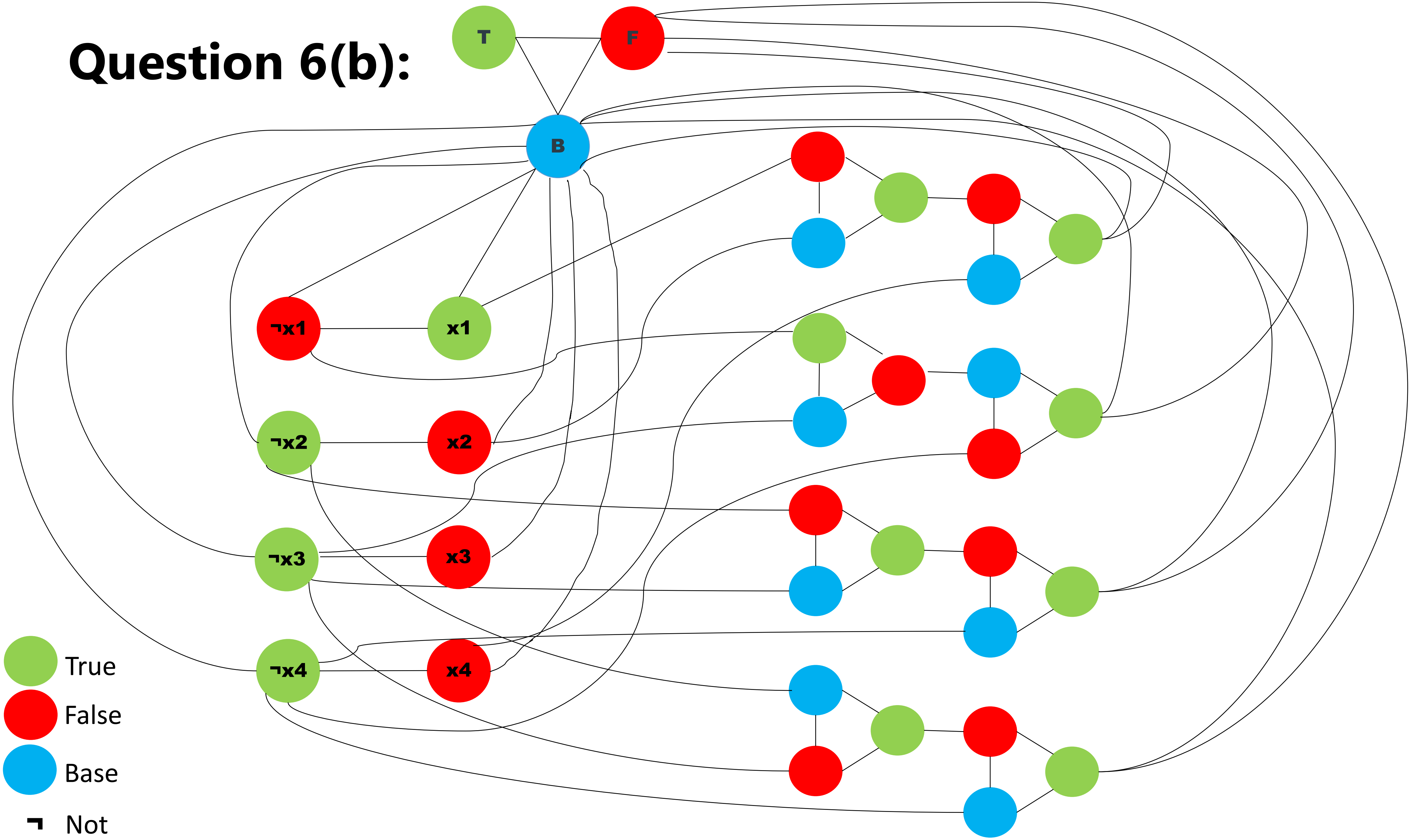


E = (x1 ∨ x2 ∨ x4) & (¬x1 ∨ ¬x3 ∨ ¬x4) & (¬x2 ∨ ¬x3 ∨ x4) & (¬x2 ∨ ¬x3 ∨ ¬x4)

Selecting Vertex Cover:



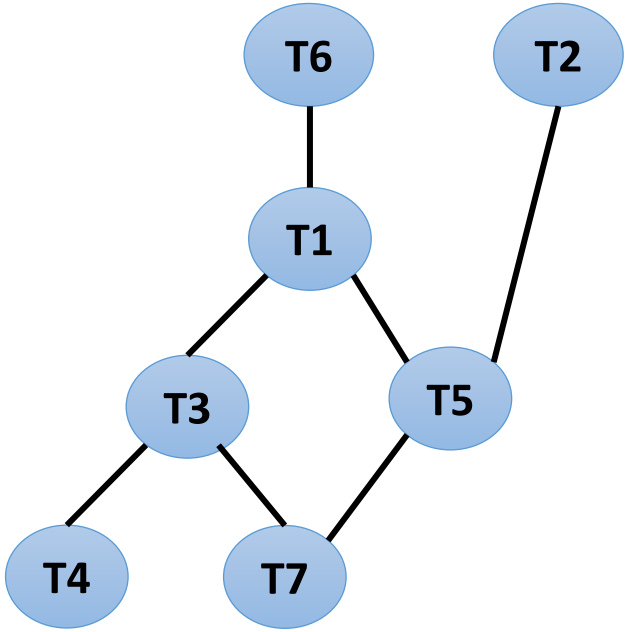
Question 6(b):



7. Task set **(T1,2), (T2,1), (T3,1), (T4,3), (T5,3), (T6,2), (T7,5)**, with partial order

T1<T3; T1<T5, T2<T5, T3<T4; T3<T7; T6<T1; T5<T7

a. Draw the graph that depicts these relationships.



b. Show the 2-processor schedule that results when the task number is the priority; a smaller task number means higher priority.

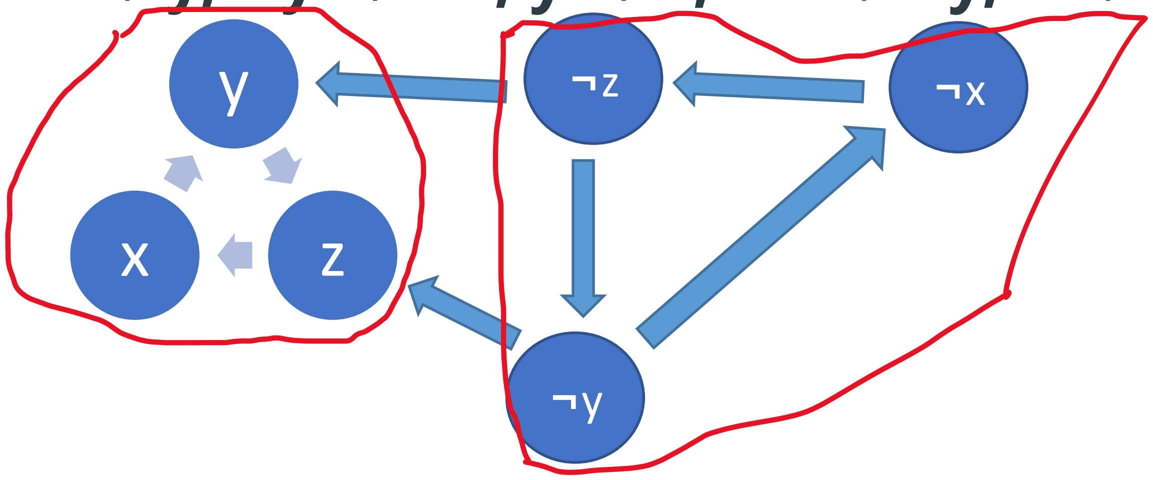
T2		T1	T1	T3	T4	T4	T4												
T6	T6			T5	T5	T5	T7	T7	T7	T7	T7								

8. Consider the following 2SAT instance.

(¬x ∨ y) (¬y ∨ z) (¬z ∨ x) (z ∨ y)

a. Draw the implication graph associated with this formula.

x → y; ¬y → ¬x; y → z; ¬z → ¬y; z → x; ¬x → ¬z; ¬z → y; ¬y → z

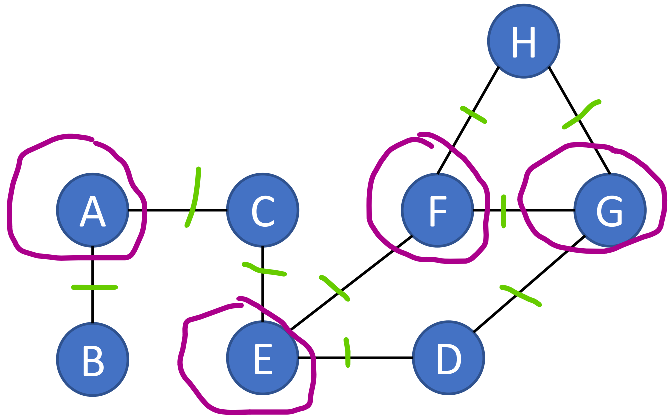


b. Draw circles around the strongly connected components (see red circles)

c. Provide a solution based on the SCCs or highlight the conflict exposed by the SCCs – the cluster with three elements has no outgoing edges, so

x = y = z = T

9. Consider the following instance of Positive Min-Ones-2SATt,
(A ∨ B) (A ∨ C) (C ∨ E) (D ∨ E) (D ∨ G) (E ∨ F) (F ∨ G) (F ∨ H) (G ∨ H)
a. Convert this instance of Positive 2SAT to a graph for which Min Vertex Cover is equivalent to the Min-Ones problem.



b. Show solution for Min Vertex Cover for (a) and correspondingly for the Positive Min-Ones-2SAT instance.

Solution: Min Cover is 4 choosing **A, E, F, G**; True assignments are is **A = E = F = G = T**

See circled nodes and covered edges with green slashes.