Logic Problems Both Easy and Hard

Last Updated: January 28th, 2026

1 Introduction

We begin this lecture by first reviewing set, function, and (propositional) logic notation. This notation will be used throughout the course and the ability to read, comprehend, and write this notation is essential for success in the course.

We then introduce the graph decision problem `Reachable` as well as the `2SAT` and `3SAT` logic decision problems. We show how to reduce the `2SAT` problem to the `Reachable` problem, thus making `2SAT` decidable in a linear number of steps. We then provide some intuition for why most computer scientists agree that `3SAT` cannot be solved in a polynomial number of steps.

2 Sets

Definition 2.1. A **set** represents a collection of items, where each item is called a **member** or **element** of the set.

List Notation the most common way to represent a set is to list the set members one-by-one, and delimit the list with curly braces.

For example,

$$\{2, 3, 5, 7, 11\}$$

uses list notation to describe the set consisting of all prime numbers that do not exceed 11. Note that the order in which the members are listed does not matter. Indeed the sets $\{2, 3, 5, 7, 11\}$ and $\{3, 11, 5, 2, 7\}$ are two different ways of writing the same set. Also, each member occurs only *once* in the set, meaning that no item can be listed more than once. Note: a **multiset** is a set for which each member may occur more than once. When using list notation to write a multiset, then we list an item a number of times that is equal to its number of occurrences in the set. For example, $\{1, 2, 2, 3, 3, 3\}$ is a multiset for which 1 occurs once, 2 occurs twice, 3 occurs thrice.

Informal List Notation uses **ellipsis** ... to indicate that a pattern is to be continued in the list, either indefinitely or up to some value.

Common Numerical Sets

Natural Numbers $\mathcal{N} = \{0, 1, 2, \ldots\}$

Integers $\mathbb{I} = \{0, \pm 1, \pm 2, \ldots\} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

Empty Set the set having no members and denoted by $\emptyset$.

Membership Symbol $x \in A$ indicates that item x is a member of set A, while $x \notin A$ means that x is not a member of A.

Containment Symbol $A \subseteq B$ means that A is a **subset** of B. In other words, every member of A is also a member of B. Note: trivially, $\emptyset \subseteq B$ for every set B, but $A \subseteq \emptyset$ is only true when $A = \emptyset$.

Proper Containment Symbol $A \subset B$ means that A is a **proper subset** of B, meaning that there is some member of B that is not a member of A. For example, $\mathcal{N} \subset \mathbb{Q}$ since there is a rational number, e.g. $\frac{1}{2}$, that is not an natural number. Similarly,

$$\mathcal{N} \subset \mathbb{I} \subset \mathbb{Q}.$$

Set Equality $A = B$ iff $A \subseteq B$ and $B \subseteq A$ are both true statements.

Set Cardinality $|A|$ denotes the number of items of A. We also refer to $|A|$ as the size of A. Note: $|\mathcal{N}| = |\mathbb{I}| = |\mathbb{Q}| = \infty$.

Powerset of a set A, denoted $\mathcal{P}(A)$, is the set consisting of all subsets of A. For example, if $A = \{1, 2, 3\}$, then

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Example 2.2. Which of the following are true statements?

The following are all true statements.

a. $63 \in \{2, 3, 5, 7, 11, \ldots, 97\}$

b. $39 \notin \{2, 3, 5, 7, 11, \ldots, 97\}$

c. $\{3\} \in \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

d. $3 \in \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

e. $\{7, 23, 59\} \subset \{2, 3, 5, 7, 11, \ldots, 97\}$

f. $\{7, 23, 59\} \not\subseteq \{2, 3, 5, 7, 11, \ldots, 97\}$

g. $\{3\} \in \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

h. $\{3\} \subset \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

i. $|\{\emptyset\}| = 0$

j. $|\emptyset| = 0$

k. $|\{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}| = 6$

□

3 Functions

Definition 3.1. A function $f : A \to B$ is a set A, a set B, and rule that assigns each member $a \in A$ to exactly one member $b \in B$. We write this as $f(a) = b$.

Some terminology related to function $f : A \to B$

Function Name f is an identifier that names the function

Domain set A is the set of **inputs** that get assigned by f

Co-Domain set B is the set of possible **outputs** to which an input can be assigned by f. Simply put, $a \in A$ is the input and $f(a) = b \in B$ is the output associated with a.

Example 3.2. Consider the sequence of numbers $24, 43, 87, 86, 68, 62, 51, 17$ and the function

$$\text{order} : A \to B,$$

where $A = \{17, 24, 43, 51, 62, 68, 86, 87\}$, $B = \{1, 2, 3, 4, 5, 6, 7, 8\}$, and order$(x)$ equals the order that x appears in the above sequence. Provide a **function table** that shows the correspondence between each input x and its corresponding output order(x) for every $x \in A$.

4 Propositional Logic

4.1 Boolean Variables and Assignments

Boolean Variable A variable is said to be **Boolean** iff its domain equals $\{0, 1\}$. We use lowercase letters, such as $x, y, z, x_1, x_2, \ldots$, etc., to denote a Boolean variable.

Assignment An **assignment** over a set of V of Boolean variables is a function $\alpha : V \to \{0, 1\}$ that assigns to each variable $x \in V$ a value in $\{0, 1\}$. We may represent α using function notation, or as a labeled tuple.

Example: for the assignment α that assigns 1 to both x_1 and x_2, and 0 to x_3, we may use function notation and write $\alpha(x_1) = 1$, $\alpha(x_2) = 1$, and $\alpha(x_3) = 0$, or we may use tuple notation and write

$$\alpha = (x_1 = 1, x_2 = 1, x_3 = 0),$$

or

$$\alpha = (1, 1, 0),$$

if the associated variables are understood.

Variable Negation If x is a Boolean variable, then $\overline{x}$ is called its **negation**.

Example: Suppose assignment α satisfies $\alpha(x_1) = 0$. Then (extending α to include negation inputs) $\alpha(\overline{x}_1) = 1$.

positive literal

Literal A **literal** is either a variable or the negation of a variable .

negative literal

Example: x_1, x_3, $\overline{x}_3$, are $\overline{x}_5$ all examples of literals.

Consistent A set R of literals is said to be **consistent** iff no variable and its negation are both in R. Otherwise, R is said to be **inconsistent**.

Example: $\{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is a consistent set, but $\{x_1, \overline{x}_2, x_4, \overline{x}_7, x_7\}$ is an inconsistent set.

Induced Assignment If $R = \{l_1, \ldots, l_n\}$ is a consistent set of literals, then α_R is called the **(partial) assignment induced by R** and is defined as $\alpha_R(x) = 1$ if $x \in R$, $\alpha_R(x) = 0$ if $\overline{x} \in R$, and $\alpha_R(x)$ is undefined for any for which $x, \overline{x} \notin R$.

Example: the assignment induced by $R = \{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is

$$\alpha_R = (x_1 = 1, x_2 = 0, x_4 = 1, x_7 = 0, x_9 = 0).$$

4.2 Logical Operations

Connective	**Symbol**	**Example**
NOT	-	"It is **not** raining outside."
AND	$\wedge$	"It is raining **and** the sidewalk is wet."
OR (Inclusive)	$\vee$	"You may board with either a passport **or** state-issued ID."
OR (Exclusive)	$\oplus$	"He was born in either Phoenix **or** Tuscon."
IF-THEN	$\rightarrow$	"**If** it rains **then** the sidewalk gets wet."
EQUIVALENCE	$\leftrightarrow$	"Passing the exam is **equivalent** to scoring at least 75 points."

4.3 Truth Tables

Each logical operation may be formally defined as a function that takes one or two Boolean values as input, and outputs a Boolean value. Below are function tables (also called truth tables) for each of the operations.

x	$\overline{x}$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

P: It's raining.

Q: The sidewalk is wet.

$P \to Q$

x	y	$x \to y$
0	0	1
0	1	1
1	0	0
1	1	1

"If x is true, then y is true"

"x implies y"

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

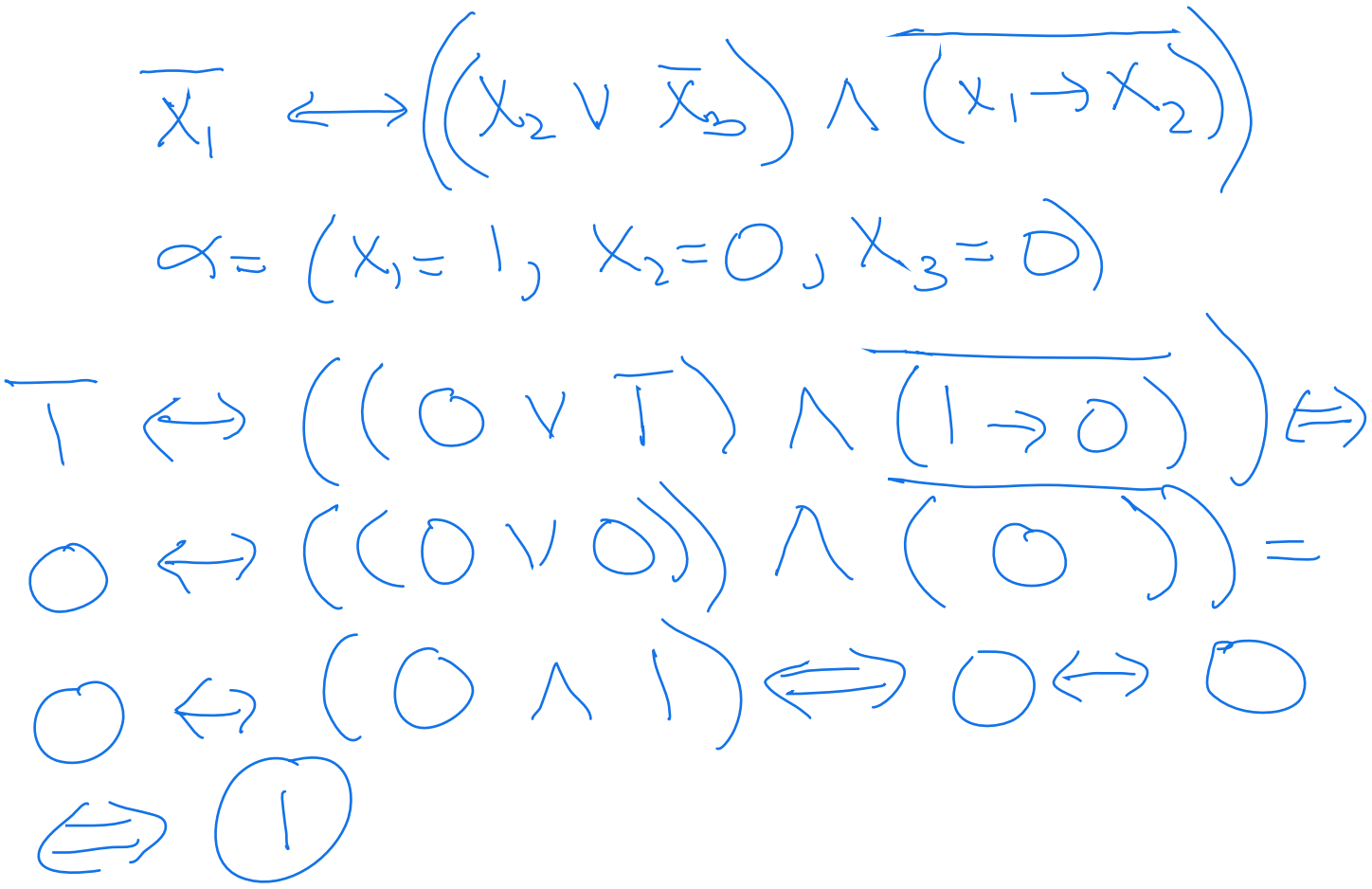
$\overline{x \leftrightarrow y} \Leftrightarrow x \oplus y$

We now comment on the $x \to y$ truth table.

1. x being true is a cause for y to be true. Hence $1 \to 1$ is true, while $1 \to 0$ is false.
2. However, if $x = 0$ then it is OK if either $y = 0$ or $y = 1$.
3. Example: if x evaluates the truth of the statement "it is raining", and y evaluates "the sidewalk is wet", then $0 \to 0$ is true since it is common for a sidewalk to be dry when it is not raining. Furthermore, $0 \to 1$ is also true, since it might not be raining, but the sprinklers could be getting the sidewalk wet.
4. Therefore, $x \to y$ means that $x = 1$ causes $y = 1$, but it may not be the only possible cause.

Definition 4.1. Similar to an arithmetic expression that uses variables, numbers, parentheses, and the set of operations $A = \{+, -, \times, \div, \text{ mod }\}$, a **Boolean expression**, also referred to as a **Boolean formula**, is the same as an arithmetic expression, but with numbers replaced with Boolean values, and A replaced with $B = \{\text{-}, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow\}$.

Example 4.2. Show some examples of both arithmetic and Boolean expressions.



5 Computational Problems

Informally, when we think of a problem, we think of a situation that needs to be resolved (i.e. solved). Moreover, in computer science we think of a computing problem as having the following properties.

Definition 5.1. A **computing problem** is a collection of situations that share a common theme, and each of which requires a solution.

- Each situation is referred to as a **problem instance**, and represents a concrete example of the general problem.
- Each problem instance can be represented by a unique word over some alphabet, meaning that no two instances map to the same word. The word may then be encoded into a binary word so that the problem instance can be stored in computer memory.

Three types of computational problems in relation to this course

Decision The solution to a problem instance is either Yes or No, equivalently True (1) or False (0). An instance x for which the solution is 1 (respectively, 0) is called a **positive instance** (respectively, **negative instance**).

Optimization The solution to a problem instance x is a number that represents the greatest (or least) quantity of some entity that is associated with x

Miscellaneous Any computation problem that is neither a decision nor an optimization problem

Example 5.2. For each of the following problems, determine if it is a decision, optimization, or miscellaneious problem.

a. An instance of the `Prime` problem is a natural number $n \geq 0$, and the problem is to decide if n is a prime number, meaning that its only natural divisors are 1 and n.

b. An instance of the `Maximum Subsequence Sum (MSS)` is a sequence (array) a of integers. The problem is to determine the greatest sum that can be made by any subsequence of a. For example, determine the maximum subsequence sum for any subsequence of

$$3, -4, 2, 5, -2, -4, 0, 2, 3, -2, 1.$$

c. An instance of `Sort` is an array a of integers. The problem is to produce another array b whose members are the members of a, but in sorted order.

d. An instance of `Fallible` is a Boolean formula F. Is there an assignment that can be made to the variables of F so that F evaluates to 0? For example, given the logical formula $F(x, y) = x \to (y \to x)$, can x and y be assigned Boolean values that force F to evaluate to 0?

Problems as Sets

Notice that every problem is given a name consisting of one or more words along with an associated acronym if necessary. We often write a problem's name to represent the *set* of all problem instances. For example,

$$f : \texttt{Sort} \to \{0, 1\}$$

is a function whose inputs are instances of problem `Sort`, and whose outputs are 0 or 1. For example, we may define $f(a) = 1$ iff integer array a is a sorted array, and $f(a) = 0$ otherwise.

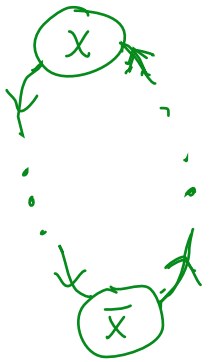
6 The `Reachability` problem

Recall that a **graph** is a pair of sets $G = (V, E)$, where V is called the **vertex set** and E is called the **edge set**, and the members of E are pairs of vertices (see the example below).

Definition 6.1. An instance of the `Reachability` decision problem is a graph $G = (V, E)$ and vertices $u, v \in V$, and the problem is decide if there is a path in G from u to v. In other words, is there a sequence of vertices

$$P = v_0, v_1, \ldots, v_k$$

for which $u = v_0$, $v = v_k$, and $(v_i, v_{i+1}) \in E$ for every $i \in \{0, 1, \ldots, i\}$?



Reachability Algorithm

Input: $G = (V, E)$, $u, v \in V$.

Output: `true` iff there is a path from u to v.

If $u = v$, then return `true`.

Initialize FIFO queue Q with u: $Q \leftarrow (u)$.

Mark u as having been reached.

While $Q \neq (\ )$

Remove vertex w from the front of Q: $Q \leftarrow Q - Q[0]$.

For each edge $(w, x) \in E$

If x is unmarked, then mark x and enter x into Q: $Q \leftarrow Q + (x)$.

If v is marked, then return `true`.

Return `false`.

Theorem 6.2. The Reachability Algorithm is correct and has the stated running time equal to $O(m+n)$.

Proof. We claim that, for all $i \geq 0$, if there is a path from u to x having length i, then x gets marked during the algorithm.

Basis step. Assume $i = 0$. Then necessarily $x = u$ which gets marked before entering the `while` loop.

Inductive step. Assume the claim is true for some $i \geq 0$. Consider a path from u to x having length $i+1$. Let w be the vertex that immediately precedes x in the path. Then there is a path from u to w having length i. By the inductive assumption, vertex w gets marked and added to Q. Thus, there will be a step in the algorithm where w is removed from Q and edge $(w, x) \in E$ will be examined. At this point x gets marked in case it has yet to be marked. □

The above inductive proof shows that, if v is reachable from u, then the algorithm returns `true`, since there is a path from u to v. Conversely, we leave it as an exercise to prove that, if a vertex gets marked during the algorithm, then that vertex must be reachable from u (hint: use induction).

Running Time. To see that the algorithm runs in linear time, notice that the `while` loop requires at most n iterations and, assuming an undirected graph, each edge (w, x) in G must be considered at most twice: once if w is removed from Q, and a second time if x is removed from Q. Thus, the total number of steps equals $O(2m+n) = O(m+n)$. □

Recall that graphs come in two different flavors: directed and undireced. Given graph $G = (V, E)$, G is said to be **directed** iff each edge $(u, v) \in E$ is oriented "from u to v", meaning that any path that traverses (u, v) must first move to u, followed by moving to v. Directed edges are often drawn using arrows, where the arrow points to v from u (see next example). On the other hand, G is said to be **undirected** if there is no such ordering placed on the vertices. In other words, a path that traverses (u, v) may either first visit u followed by v, or vice versa. Thus undirected graphs have bidirectional edges with no arrows. Finally, an undirected graph is said to be **simple** provided there is at most one edge between any two vertices, and there are not self loops, i.e. edges of the form (u, u).

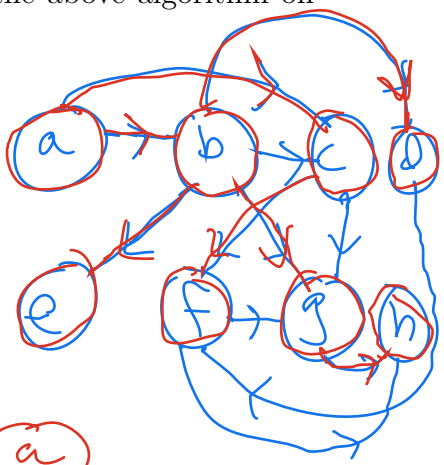
Example 6.3. Show the contents of the queue Q during the execution of the above algorithm on the directed graph $G = (V, E)$, where

$$V = \{a, b, c, d, e, f, g, h\}$$

and the edges are given by

$$E = \{(a, b), (a, c), (b, c), (b, d), (b, e), (b, g), (c, g), (c, f),$$
$$(d, f), (f, g), (f, h), (g, h)\}.$$

Decide if h is reachable from a.



$Q = a$

$Q = b, c$

$Q = c, d, e, g$

$Q = d, e, g, f$

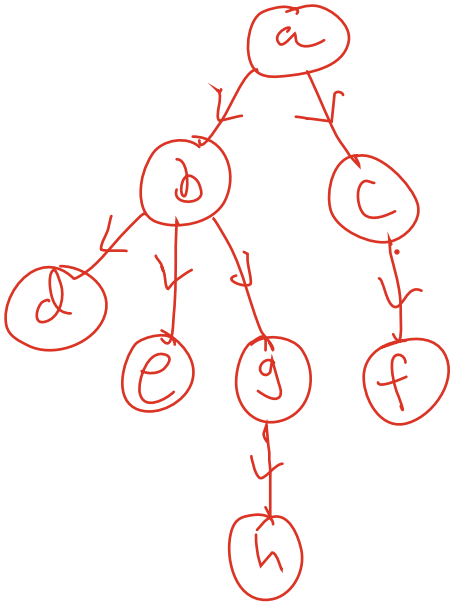
$Q = e, g, f$

$Q = g, f$

$Q = f, h$

$Q = h$

$Q = \emptyset$



7 The 2SAT decision problem

Definition 7.1. A **binary disjunctive clause** is a Boolean formula of the form

$$l_1 \vee l_2,$$

where l_1 and l_2 are literals. The clause evaluates to 1 in case either l_1 or l_2 (or both) is assigned 1.

Definition 7.2. An instance of the 2SAT decision problem consists of a set $\mathcal{C}$ of binary disjunctive clauses. The problem is to decide if there is an assignment α over the variables in $\mathcal{C}$, such that every clause $(l_1 \vee l_2)$ in $\mathcal{C}$ evaluates to 1 under α. If such an assignment α exists, then it is said to be a **satisfying assignment** and we say $\mathcal{C}$ is **satisfiable**. Otherwise, $\mathcal{C}$ is said to be **unsatisfiable**. Finally, the 2SAT decision problem is the problem of deciding whether a set $\mathcal{C}$ of clauses is sastisfiable.

Simplified clause notation. In what follows, we often simplify the clause notation by writing each clause $(l_1 \vee l_2)$ as (l_1, l_2).

Example 7.3. Provide a satisfying assignment for

$$\mathcal{C} = \{(x_2, \overline{x}_3), (x_1, \overline{x}_2), (x_3, x_4), (\overline{x}_2, \overline{x}_3), (\overline{x}_1, \overline{x}_4)\}.$$

$\alpha = (x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1)$

$\therefore$ α satisfies $\mathcal{C}$

and $\mathcal{C}$ is a positive instance of 2SAT

We now demonstrate how to "reduce" `2SAT` to `Reachability`, meaning that we can solve an instance of `2SAT` by constructing a graph and examining the "reachability sets" for the vertices of the graph.

Definition 7.4. Let $\mathcal{C}$ be an instance of `2SAT`, and defined over the variables $x_1, x_2, \ldots, x_n$. Then the **implication graph** of $\mathcal{C}$ is defined as the directed graph $G_{\mathcal{C}} = (V, E)$, where

$$V = \{x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$$

and each clause $(l_1 \vee l_2)$ produces the two directed edges $(\overline{l_1}, l_2), (\overline{l_2}, l_1) \in E$.

The idea behind the two edges formed from clause $c = (l_1 \vee l_2)$ is that c is logically equivalent to both the implication $\overline{l_1} \to l_2$ and its **contrapositive** $\overline{l_2} \to l_1$. Thus, for each edge (l_1, l_2) of $G_{\mathcal{C}}$, when l_1 is assumed true, then l_2 must also be true. This is so because (l_1, l_2) corresponds with the clause $(\bar{l}_1 \vee l_2)$ and the truth of l_1 forces the truth of l_2, since, according to the clause, either l_1 must be false or l_2 must be true.

$\overline{l_2} \to \overline{\overline{l_1}} \Leftrightarrow l_1$

l_1	l_2	$\overline{l_1}$	$\overline{l_2}$	$l_1 \vee l_2$	$\overline{l_1} \to l_2$	$\overline{l_2} \to l_1$
0	0	1	1	0	0	0
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	0	1	1	1

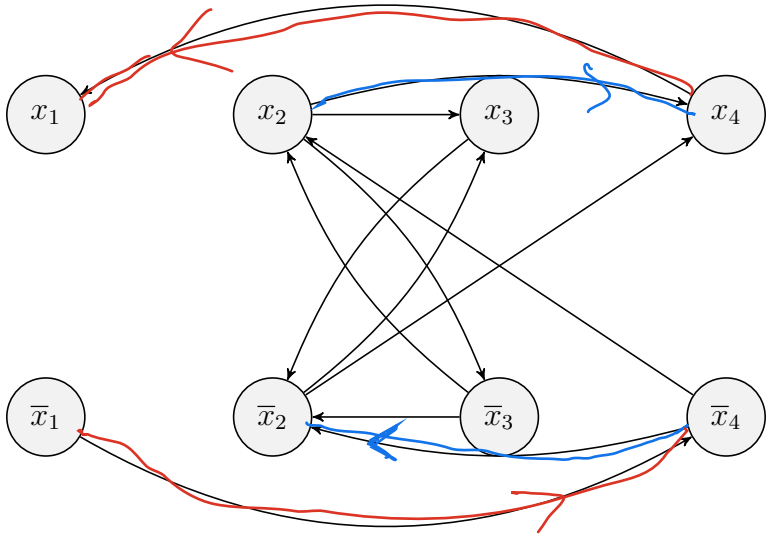
$P \to Q$ contrapositive: $\overline{Q} \to \overline{P}$

Example 7.5. Verify that $l_1 \vee l_2$ is logically equivalent to both $\overline{l_1} \to l_2$ and $\overline{l_2} \to l_1$, meaning that all three have the same truth table.

Solution. See previous page

Example 7.6. Draw the implication graph for the set $\mathcal{C}$ of clauses listed in the following table.

Clause	Implication	Contrapositive
$(\overline{x}_2, x_4)$	$x_2 \to x_4$	$\overline{x}_4 \to \overline{x}_2$
$(\overline{x}_2, \overline{x}_3)$	$x_2 \to \overline{x}_3$	$x_3 \to \overline{x}_2$
$(\overline{x}_2, x_3)$	$x_2 \to x_3$	$\overline{x}_3 \to \overline{x}_2$
(x_2, x_3)	$\overline{x}_2 \to x_3$	$\overline{x}_3 \to x_2$
(x_2, x_4)	$\overline{x}_2 \to x_4$	$\overline{x}_4 \to x_2$
$(x_1, \overline{x}_4)$	$\overline{x}_1 \to \overline{x}_4$	$x_4 \to x_1$



Implication Graph $G_{\mathcal{C}}$

Given the correspondence of a 2SAT instance $\mathcal{C}$ with an implication graph $G_{\mathcal{C}}$ whose number of vertices is twice the number n of variables, and whose number of edges is twice the number m of clauses, it seems appropriate to let $m = |\mathcal{C}|$ and n represent the size parameters for 2SAT.

Proposition 7.7. Given implication graph $G_{\mathcal{C}}$ and path

$$P = l_1, \ldots, l_n,$$

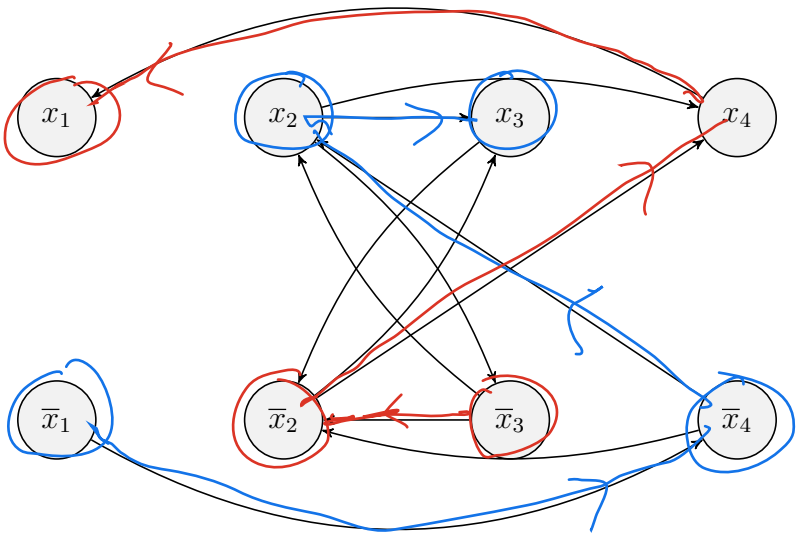
in $G_{\mathcal{C}}$, there is another path, called the **contrapositive** of P:

$$\overline{P} = \bar{l}_n, \ldots, \bar{l}_1.$$

Proof. If (x, y) is an edge of $G_{\mathcal{C}}$, then so is its contrapositive $(\overline{y}, \overline{x})$, since both are logically equivalent to $\overline{x} \vee y$. Thus, every edge in a path $P = l_1, l_2, \ldots, l_{n-1}, l_n$ corresponds with its contrapositive in the path $\overline{P} = \bar{l}_n, \bar{l}_{n-1}, \ldots, \bar{l}_2, \bar{l}_1$, and thus both P and $\overline{P}$ are paths of $G_{\mathcal{C}}$. □

Example 7.8. Given the path $P = \overline{x}_1, \overline{x}_4, x_2, x_3$ with respect to the implication graph $G_{\mathcal{C}}$ below, verify that $\overline{P}$ is also in $G_{\mathcal{C}}$.

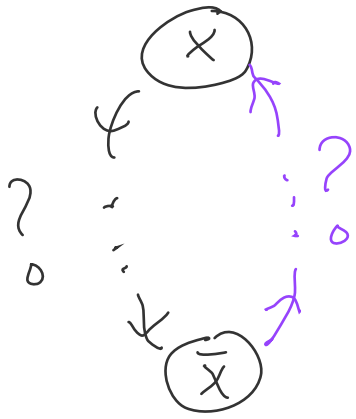
$\overline{P} = \overline{x}_3, \overline{x}_2, \overline{x}_4, \overline{x}_1 = \overline{x}_3, \overline{x}_2, x_4, x_1.$



Implication Graph $G_{\mathcal{C}}$

Theorem 7.9. `2SAT` instance $\mathcal{C}$ is satisfiable iff every cycle in $G_{\mathcal{C}}$ is **consistent**, meaning that, if C is any cycle in $G_{\mathcal{C}}$, then the set of literals that comprises the vertices of C is consistent.

Before proving Theorem 7.9, we use it to provide an initial algorithm showing that `2SAT` can be polynomial-time reduced to `Reachability` by calling the `Reachability` algorithm at most $2n$ times. The algorithm uses the observation that $G_{\mathcal{C}}$ has only consistent cycles iff, for every variable x, either x is not reachable from $\overline{x}$, or $\overline{x}$ is not reachable from x (or both). For each x, this can be done with two queries to a `Reachability`-oracle.



2SAT Algorithm

Input: `2SAT` instance $\mathcal{C}$.

Ouput: `true` iff $\mathcal{C}$ is satisfiable.

Construct $G_{\mathcal{C}}$.

For each $x \in \text{var}(\mathcal{C})$,

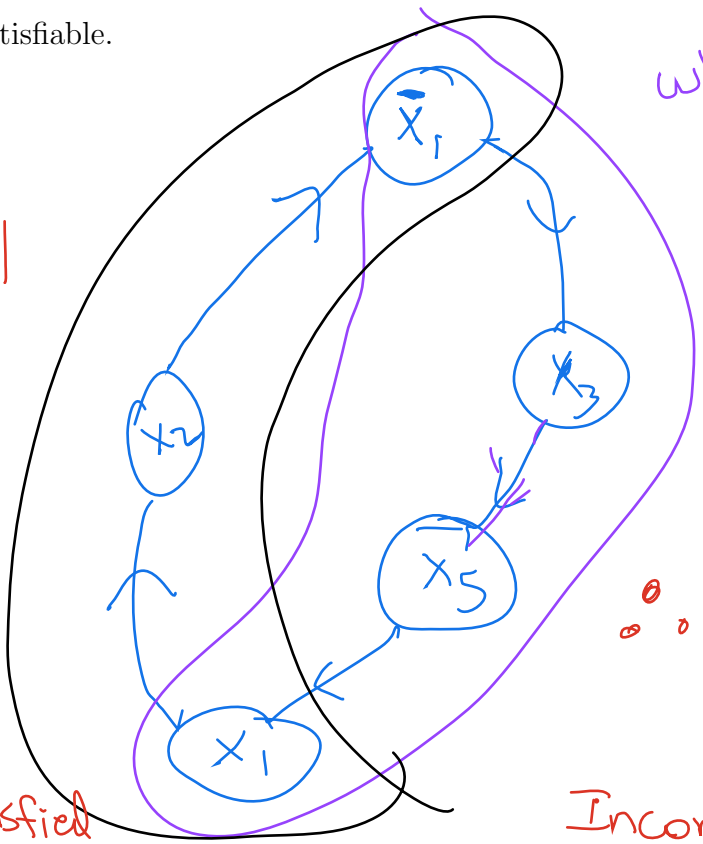
If `reachable`$(G_{\mathcal{C}}, x, \overline{x})$ and `reachable`$(G_{\mathcal{C}}, \overline{x}, x)$, then return `false`.

Return `true`.

Example 7.10. Consider a 2SAT instance $\mathcal{C}$ for which $G_{\mathcal{C}}$ has the inconsistent cycle

$$C = \overline{x}_1, x_3, \overline{x}_5, x_1, x_2, \overline{x}_1.$$

Verify that $\mathcal{C}$ is unsatisfiable.



Similarly,
if $\alpha'(x_1) = 1$
then
$(\overline{x}_1 \vee x_2) = C_1'$
$(\overline{x}_2 \vee \overline{x}_1) = C_2'$
C_2' is not satisfied

$\overline{x}_1 \to x_3$
what if $\alpha(x_1) = 0$
$C_1 = (x_1 \vee x_3)$
$C_2 = (\overline{x}_3 \vee \overline{x}_5)$
$C_3 = (x_5 \vee x_1)$
$\therefore$ C_3 is not satisfied

Incons. Cycle $\to$ unsat
unsat $\to$ Incons. cycle

We can generalize the previous example by stating the following proposition.

Proposition 7.11. Given a 2SAT instance $\mathcal{C}$ that includes variable x, the following statements are true.

1. If there is a path from x to $\overline{x}$ then no assignment that assigns $x = 1$ can satisfy $\mathcal{C}$.
2. If there is a path from $\overline{x}$ to x then no assignment that assigns $x = 0$ can satisfy $\mathcal{C}$.

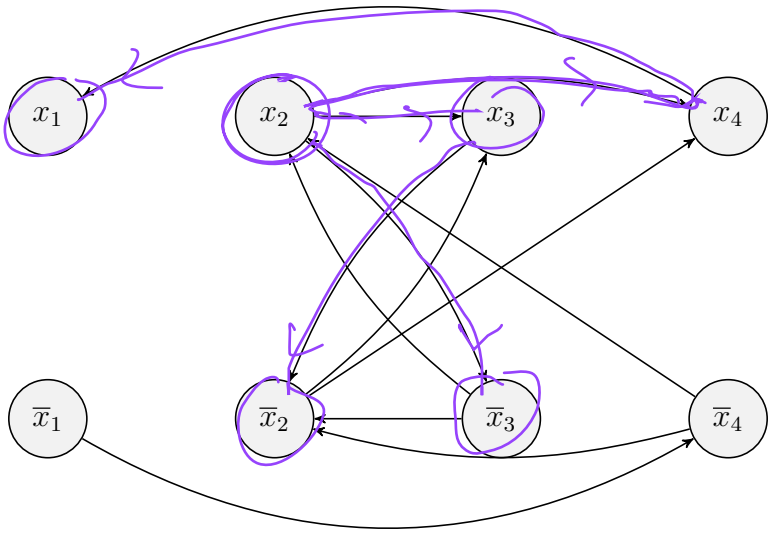
Proposition 7.14 below provides the key to finding a satisfying assignment for 2SAT instance $\mathcal{C}$ in case all of $G_{\mathcal{C}}$'s cycles are consistent. It relies on the notion of a *reachability set* for a graph vertex.

Definition 7.12. Let $G = (V, E)$ be a graph and $v \in V$ a vertex of G. Then the **reachability set** of v in G is the set of all vertices that can be reached by v along some path (regardless of its length).

Example 7.13. Verify that the reachability set for vertex x_2 of the implication graph in Example 7.6 is equal to

$$R_{x_2} = \{x_2, x_3, \overline{x}_3, x_4, \overline{x}_2, x_1\}.$$

Is R_{x_2} a consistent or inconsistent set of literals?



Proposition 7.14. Given 2SAT instance $\mathcal{C}$, implication graph $G_{\mathcal{C}}$, and vertex/literal l, if R_l, the following statements are true.

1. If R_l is an inconsistent set of literals, then no assignment that assigns $l = 1$ can satisfy $\mathcal{C}$.
2. If R_l is a consistent set of literals, then assignment α_{R_l} satisfies every clause in $\mathcal{C}$ that depends on at least one variable assigned by α_{R_l}.

Proof of Statement 1.

A. Since R_l is inconsistent, There is a variable x for which both x and $\overline{x}$ belong to R_l.

B. Let $P = l, l_1, \ldots, l_r, x$ be a path from l to x.

C. Let $Q = l, l'_1, \ldots, l'_s, \overline{x}$ be a path from l to $\overline{x}$.

D. Then $P \circ \overline{Q} = l, l_1, \ldots, l_r, x, \overline{l'_s}, \ldots, \overline{l'_1}, \overline{l}$ is a path from l to $\overline{l}$.

E. Therefore, by Propostion 7.11, no assignment that assigns $l = 1$ can satisfy $\mathcal{C}$. □

Proof of Statement 2.

A. Without loss of generality, we assume that x is a variable that is reachable from l (a similar argument can be made by assuming $\overline{x}$ is reachable from l).

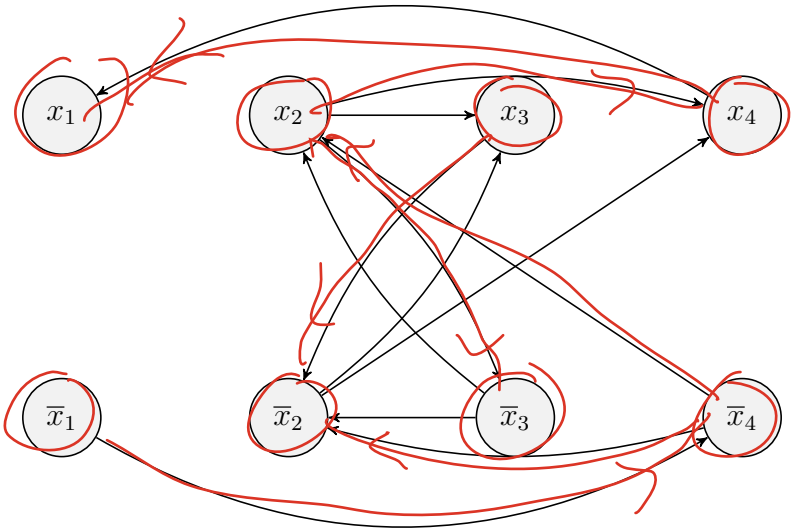
B. Consider any clause $c = (x \vee l')$, where l' is some literal. Then, since $x \in R_l$, $\alpha_{R_l}(x) = 1$ and c is satisfied.

C. Now consider any clause $c = (\overline{x} \vee l')$, where again l' is some literal. Notice that (x, l') is an edge of $G_{\mathcal{C}}$. Hence, since x is reachable from l, l' is also reachable from l. Thus, since $l' \in R_l$, $\alpha_{R_l}(l') = 1$ and c is satisfied.

D. Therefore, so long as a clause c has either a variable or its negation that is reachable from l, then c will be satisfied by α_{R_l} and α_{R_l} represents a **partial satisfying assignment** for $\mathcal{C}$. □

Example 7.15. For the implication graph below, verify that the reachability set for vertex $\overline{x}_1$ contains both x_2 and $\overline{x}_2$, and so $R_{\overline{x}_1}$ is an inconsistent reachability set which means that no assignment with $x_1 = 0$ can satisfy $\mathcal{C}$.

$R(\overline{x}_1) = \{\overline{x}_1, \overline{x}_4, x_1, x_2, \overline{x}_2, x_3, \overline{x}_3, x_4\}$

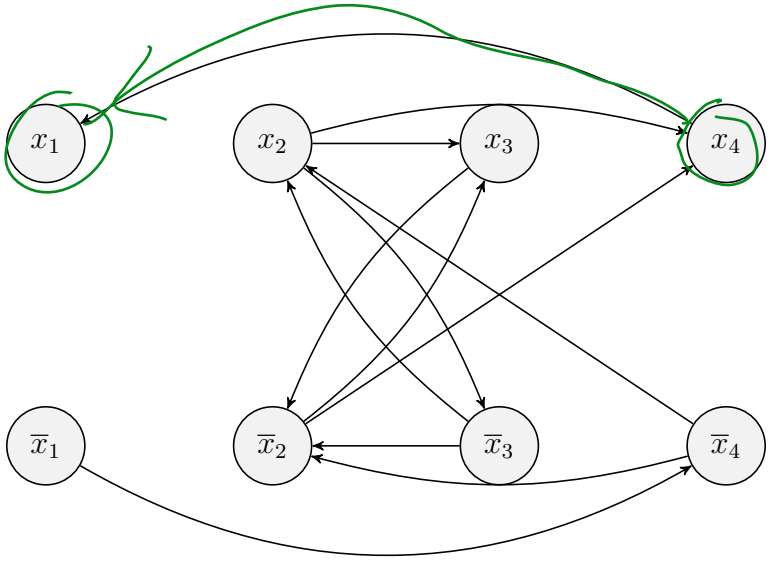


Example 7.16. Verify that the reachability set for vertex x_4 of the implication graph in Example 7.6 is equal to the consistent set

$$R_{x_4} = \{x_4, x_1\},$$

and show that $\alpha_{R_{x_4}}$ satisfies all clauses of $\mathcal{C}$ that use one of the variables from R_{x_4}.

Clause	Implication	Contrapositive
$(\overline{x}_2, x_4)$	$x_2 \to x_4$	$\overline{x}_4 \to \overline{x}_2$
$(\overline{x}_2, \overline{x}_3)$	$x_2 \to \overline{x}_3$	$x_3 \to \overline{x}_2$
$(\overline{x}_2, x_3)$	$x_2 \to x_3$	$\overline{x}_3 \to \overline{x}_2$
(x_2, x_3)	$\overline{x}_2 \to x_3$	$\overline{x}_3 \to x_2$
(x_2, x_4)	$\overline{x}_2 \to x_4$	$\overline{x}_4 \to x_2$
$(x_1, \overline{x}_4)$	$\overline{x}_1 \to \overline{x}_4$	$x_4 \to x_1$



Proof of Theorem 7.9. The statement of the theorem is of the form $P \leftrightarrow Q$, where

1. P stands for "$\mathcal{C}$ is satisfiable"
2. Q stands "$G_{\mathcal{C}}$ has only consistent cycles".
3. Thus, We can thus prove the two mathematical statements: $P \to Q$ and $Q \to P$.

Proving $P \to Q$

1. We use an indirect proof by proving the contrapositive of $P \to Q$, namely $\overline{Q} \to \overline{P}$.
2. Assume $\overline{Q}$: $G_{\mathcal{C}}$ has at least one inconsistent cycle.
3. Then there is a variable x in the cycle for which there is a path from x to $\overline{x}$ as well as a path from $\overline{x}$ to x.
4. Then both R_x and $R_{\overline{x}}$ are inconsistent reachcability sets.
5. Hence, by Proposition 7.14, there is no truth assignment that will satisfy $\mathcal{C}$.
6. Therefore, $\overline{P}$ is true namely that $\mathcal{C}$ is unsatisfiable. □

Proof of Theorem 7.9 Continued.

Proving $Q \to P$

1. Assume Q: $G_{\mathcal{C}}$ has only consistent cycles.
2. To prove P : "$\mathcal{C}$ is satisfiable" we use mathematical induction on the number of variables n that appear in one or more of the clauses of $\mathcal{C}$.
3. **Basis Step.** $n = 0$, i.e. $\mathcal{C} = \emptyset$. Then $\mathcal{C}$ is satisfiable. Why? It helps to reframe the definition of satisfiability as "there is some assignment α for which no clause is unsatisified by α". This definition is equivalent to the one given Definition 7.2 when $\mathcal{C} \neq \emptyset$. Moreover, using this restated definition, $\mathcal{C} = \emptyset$ implies that $\mathcal{C}$ is satisfiable since the empty assignment $\alpha = (\ )$ is a variable assignment for which no clause of $\mathcal{C}$ is unsatisfied by α (because $\mathcal{C}$ has no clauses!).
4. **Induction Step.** Assume that any `2SAT` instance having $n - 1$ or fewer variables and only consistent cycles in its implication graph is satisfiable, for some $n \geq 1$. Let $\mathcal{C}$ be a `2SAT` instance with n variables and only consistent cycles. We show that $\mathcal{C}$ must also be satisfiable.
5. Let x be a variable of $\mathcal{C}$. Then by assumption is must be true that either there is no path from x to $\overline{x}$ in $G_{\mathcal{C}}$ or there is no path from $\overline{x}$ to x. Without loss of generality, assume that there is no path from x to $\overline{x}$.
6. Then R_x must be a consistent set of literals. Otherwise, as was shown in the proof of Proposition 7.14, $\overline{x}$ would be a member of R_x which we've assumed is not the case.
7. Moreover, Proposition 7.14 also implies that α_{R_x} satisfies every clause that has a variable that gets assigned by α_{R_x}, including x.
8. Let C_{R_x} denote the set of clauses satisfied by α_{R_x} and consider the new `2SAT` instance $\mathcal{C}' = \mathcal{C} - C_{R_x}$ that is the result of removing the clauses in C_{R_l} from $\mathcal{C}$.
9. Then $\mathcal{C}'$ has fewer than n variables, and since $G'_{\mathcal{C}}$ is a subgraph of $G_{\mathcal{C}}$, it follows that $G'_{\mathcal{C}}$ has only consistent cycles.
10. Hence, by the inductive assumption, $\mathcal{C}'$ is satisfiable.
11. Let α' denote a satisfying assignment for $\mathcal{C}'$, then $\alpha' \cup \alpha_{R_x}$ satisfies $\mathcal{C}$.
12. Therefore $\mathcal{C}$ is satisfiable. □

The proof of Theorem 7.9 suggests the following recursive algorithm for determining the satisfiability of `2SAT` instance $\mathcal{C}$. The algorithm returns a non-empty satisfying assignment if $\mathcal{C}$ is satisfiable, and returns $\emptyset$ otherwise. In the algorithm we assume that the variables appearing in the root problem are indexed as $x_1, \ldots, x_n$, for some $n \geq 1$.

Improved 2SAT Algorithm

Name: `improved_2sat`

Input: `2SAT` instance $\mathcal{C}$ and a pointer α to an assignment (initially, $\emptyset$).

Output: `true` iff $\mathcal{C}$ is satisfiable.

Side Effect: if $\mathcal{C}$ is satisfiable, then α points to a sat assignment. Otherwise, $\alpha \leftarrow \emptyset$.

//Base Case:

If $\mathcal{C} = \emptyset$, return `true`. //an empty set of clauses is considered satisfied

//Recursive case:

Construct $G_{\mathcal{C}}$.

Let i be the least index for which x_i is a variable of $\mathcal{C}$.

$S \leftarrow \emptyset$ //S is the desired set of consistent literals and initialized as empty

If R_{x_i} is a consistent reachability set, then $S \leftarrow R_{x_i}$.

If $S = \emptyset$ and $R_{\overline{x}_i}$ is a consistent reachability set, then $S \leftarrow R_{\overline{x}_i}$.

If $S = \emptyset$

> $\alpha \leftarrow \emptyset$.
>
> Return `false`.

//$S \neq \emptyset$

Update α: $\alpha \leftarrow \alpha \cup \alpha_S$.

$\mathcal{C}' \leftarrow \mathcal{C} - C_S$, where C_S denotes all clauses satisfied by α_S.

Return `improved_2sat`$(\mathcal{C}', \alpha)$.

Notice that the algorithm requires $O(m + n)$ steps since every edge of $G_{\mathcal{C}}$ is traversed at most once when using the reachability algorithm to compute either R_{x_i} or $R_{\overline{x}_i}$.

Example 7.17. Use the improved 2SAT algorithm to determine a satisfying assignment for

$$\mathcal{C} = \{(x_2, \overline{x}_3), (x_1, \overline{x}_2), (x_3, x_4), (\overline{x}_2, \overline{x}_3), (\overline{x}_1, \overline{x}_4), (x_5, x_6), (\overline{x}_5, \overline{x}_6), (\overline{x}_1, x_6)\}.$$

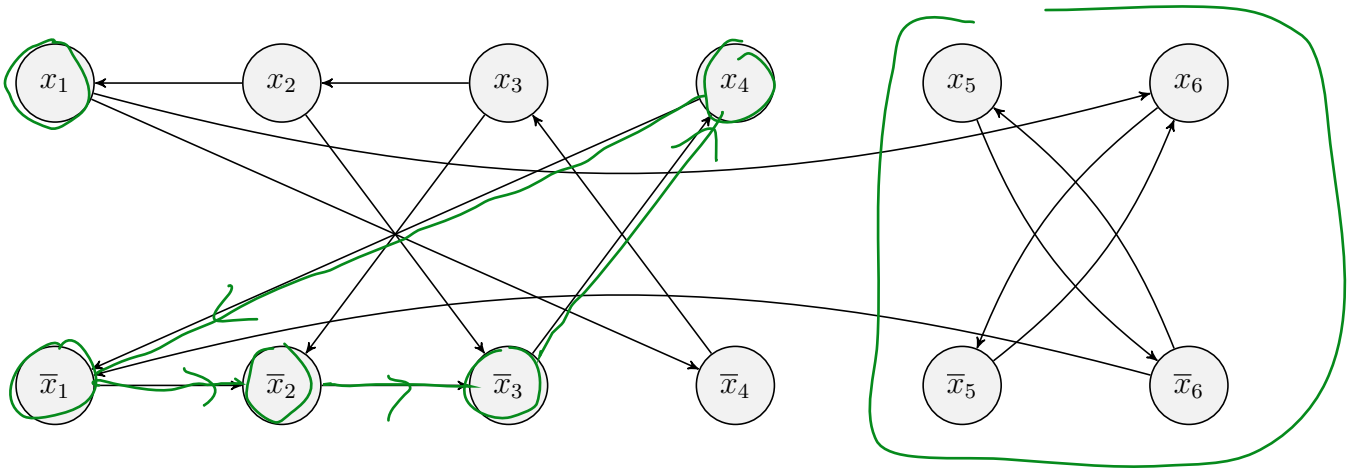
Start off by choosing $l = x_1$.

Solution.

1. Compute the edges for $G_{\mathcal{C}}$.

Clause	Edges
$(x_2, \overline{x}_3)$	$\overline{x}_2 \to \overline{x}_3,\ x_3 \to x_2$
$(x_1, \overline{x}_2)$	$\overline{x}_1 \to \overline{x}_2,\ x_2 \to x_1$
(x_3, x_4)	$\overline{x}_3 \to x_4,\ \overline{x}_4 \to x_3$
$(\overline{x}_2, \overline{x}_3)$	$x_2 \to \overline{x}_3,\ x_3 \to \overline{x}_2$
$(\overline{x}_1, \overline{x}_4)$	$x_1 \to \overline{x}_4,\ x_4 \to \overline{x}_1$
(x_5, x_6)	$x_5 \to x_6,\ \overline{x}_6 \to x_5$
$(\overline{x}_5, \overline{x}_6)$	$x_5 \to \overline{x}_6,\ x_6 \to \overline{x}_5$
$(\overline{x}_1, x_6)$	$x_1 \to x_6,\ \overline{x}_6 \to \overline{x}_1$

2. Draw the implication graph



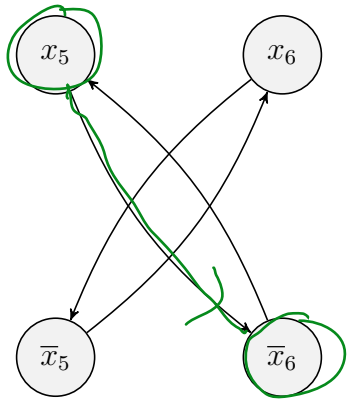
3. Compute $R_{x_1} = \{x_1, \overline{x}_1, x_2, \overline{x}_2, x_3, \overline{x}_3, x_4, \overline{x}_4, \overline{x}_5, x_6\}$ which is inconsistent.

4. Compute $R_{\overline{x}_1} = \{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4\}$ which is consistent. Verify that

$$\alpha_{R_{\overline{x}_1}} = (x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1)$$

satisfies all clauses that involve variables x_1, x_2, x_3, x_4.

5. Draw the reduced implication graph $G_{\mathcal{C}'}$ where $\mathcal{C}' = \{(x_5, x_6), (\overline{x}_5, \overline{x}_6)\}$.



6. Compute $R_{x_5} = \{x_5, \overline{x}_6\}$ which is consistent. Verify that $\alpha_{R_{x_5}} = (x_5 = 1, x_6 = 0)$ satisfies both clauses in $\mathcal{C}'$.

7. Final satisfying assignment:

$$\alpha = \alpha_{R_{\overline{x}_1}} \cup \alpha_{R_{x_5}} = (x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 1, x_6 = 0).$$

8 The 3SAT Logic Problem

In this Section we introduce the 3SAT logic problem which will which represents one of the more important problems in all of Computer Science.

Definition 8.1. A **ternary disjunctive clause** is a Boolean formula of the form

$$l_1 \vee l_2 \vee l_3,$$

where l_1, l_2, and, l_3 are literals. The clause evaluates to 1 in case at least one of l_1, l_2, or l_3 is assigned 1. In this case we say the clause is **satisfied**. Otherwise it is **unsatisfied**.

Definition 8.2. An instance of the 3SAT decision problem consists of a set $\mathcal{C}$ of ternary disjunctive clauses. The problem is to decide if there is an assignment α over the variables in $\mathcal{C}$, such that every clause $(l_1 \vee l_2 \vee l_3)$ in $\mathcal{C}$ evaluates to 1 under α. If such an assignment α exists, then it is said to be a **satisfying assignment** and we say $\mathcal{C}$ is **satisfiable**. Otherwise, $\mathcal{C}$ is said to be **unsatisfiable**. Finally, the 3SAT decision problem is the problem of deciding whether a set $\mathcal{C}$ of ternary clauses is sastisfiable.

Simplified clause notation. In what follows, we often simplify the clause notation by writing each clause $(l_1 \vee l_2 \vee l_3)$ as (l_1, l_2, l_3).

Example 8.3. Provide a satisfying assignment for

$$\mathcal{C} = \{(x_1, x_2, x_3), (\overline{x}_2, x_3, \overline{x}_4), (x_1, x_2, \overline{x}_4), (\overline{x}_1, \overline{x}_3, \overline{x}_4), (\overline{x}_1, x_2, x_4), (\overline{x}_2, x_3, x_4), (x_1, \overline{x}_3, x_4),$$

$$(\overline{x}_2, \overline{x}_3, x_4), (\overline{x}_2, \overline{x}_3, \overline{x}_4)\}.$$

Why is 3SAT Hard to Solve?

1. At this writing the best known algorithm for solving 3SAT requires approximately $O(1.33^n)$ steps meaning that it has exponential growth and thus has very limited applicability to problems whose number of variables n is in the tens to hundreds.

2. Although there are many heuristics for solving instances of 3SAT, each has a worst-case performance that is either provably inefficient or is highly suspected of being inefficient.

3. There are other computing problems that share a similar phenomenon: easy to solve for the "case of 2" but hard to solve for the "case of 3". For example, there is an efficient $O(n^2)$ algorithm for finding a matching in a bipartite graph which may be viewed as a collection of 2-tuples, yet there is no known algorithm for finding a "3-dimensional matching" in a collection of 3-tuples.

4. When moving from binary disjunctions to ternary disjunctions, when a Boolean variable x is assigned a value, it is not certain that the assigned value is correct. Moreover, several other variable assignments may be necessary before discovering that x was incorrectly assigned.

5. Contrast this with 2SAT where a variable can be correctly assigned by considering its reachability sets which both may be computed in $O(m+n)$ steps. Assuming the 2SAT instance is satisfiable, and when computing either one or both of its reachability sets, its assigned value (along with other variables in the reachability set) becomes known.

6. 3SAT has a **phase transition number**, approximately 4.267, for randomly generated instances and is defined as the critical clause-to-variable ratio $\rho = m/n$ where randomly generated instances having n variables and m clauses transition from almost certainly being satisfiable to almost certainly being unsatisfiable.

Other Efficiently Solvable Logic Problems

1. **Systems of Linear Equations** for which each formula has the form $x_{i_1} \oplus \cdots \oplus x_{i_j}$. A set of these formulas can be solve using Gaussian elimination.

2. **Horn Logic** for which each formula has the form of either i) a positive literal x_i, ii) the form $x_i \rightarrow x_{j_1} \wedge \cdots \wedge x_{j_k}$, or iii)
$$\overline{(x_{i_1} \wedge \cdots \wedge x_{i_k})}.$$
Horn logic problems can be solved in a linear number of steps.

Core Exercises

1. Which of the following statements are true? Explain.

   a. $\{16, 128, 256\} \subseteq \{1, 4, 16, 64, \ldots\}$

   b. $c \in \mathcal{P}(\{a, b, c, d, e\})$

   c. $|\mathcal{P}(\{a, b, c, d, e\})| = 32$

   d. $37 \notin \{5, 7, 10, 14, 19, \ldots\}$

2. Consider the function $f : \mathcal{N} \to \mathcal{P}(\mathcal{N})$, where $f(n)$ equals the set of prime factors of n, i.e. those prime numbers that divide evenly into n. Compute, $f(1)$, $f(28)$, and $f(2025)$.

3. Recall that the union

   $$A_1 \cup A_2 \cup \cdots \cup A_k$$

   of k sets $A_1, \ldots, A_k$ is the set A where $x \in A$ iff x is a member of A_i for at least one $i \in \{1, \ldots, k\}$. For example, if $A_1 = \{6, 11, 12, 19\}$, $A_2 = \{1, 2, 10, 12\}$, and $A_3 = \{2, 3, 4, 14\}$, then

   $$A_1 \cup A_2 \cup A_3 = \{1, 2, 3, 4, 6, 10, 11, 12, 14, 19\}.$$

   Now consider the `Set Cover` decision problem which is a triple $(\mathcal{S}, m, k)$, where $\mathcal{S} = \{S_1, \ldots, S_n\}$ is a collection of n subsets, where $S_i \subseteq \{1, \ldots, m\}$, for each $i = 1, \ldots, n$, and a nonnegative integer k. The problem is to decide if there are k subsets $S_{i_1}, \ldots, S_{i_k}$ for which

   $$S_{i_1} \cup \cdots \cup S_{i_k} = \{1, \ldots, m\}.$$

   Provide a collection $\mathcal{S}$ of $n = 9$ subsets of $\{1, \ldots, 15\}$ for which $(\mathcal{S}, m = 15, k = 5)$ is a positive instance of `Set Cover`, but $(\mathcal{S}, m = 15, k = 4)$ is a negative instance.

4. Find a satisfying assignment for the set of clauses

   $$\mathcal{C} = \{(x_1, x_2), (\overline{x}_3, \overline{x}_4), (x_3, \overline{x}_5), (x_2, x_5), (\overline{x}_2, x_3), (\overline{x}_1, \overline{x}_4), (\overline{x}_1, \overline{x}_5), (\overline{x}_2, x_5)\}.$$

5. Given `2SAT` instance $\mathcal{C}$, if $G_{\mathcal{C}}$ has path $P = x_2, \overline{x}_3, x_5, \overline{x}_1, \overline{x}_7$, then provide $\overline{P}$.

6. Given `2SAT` instance $\mathcal{C}$, if $G_{\mathcal{C}}$ has the inconsistent cycle $C = x_3, \overline{x}_4, x_5, x_1, x_4, \overline{x}_2, x_3$, then provide the subset $\mathcal{C}' \subseteq \mathcal{C}$ of clauses that is associated with this cycle. Which clauses in $\mathcal{C}'$ prevent a satisfying assignment from assigning $x_4 = 0$? Which clauses in $\mathcal{C}'$ prevent a satisfying assignment from assigning $x_4 = 1$? Conclude that $\mathcal{C}'$ (and $\mathcal{C}$ itself) is unsatisfiable.

7. For `2SAT` instance $\mathcal{C}$, suppose you make the query `reachable`$(G_{\mathcal{C}}, x_3, \overline{x}_3)$ to a `Reachability` oracle who answers the query with "yes". Assuming $\mathcal{C}$ is satisfiable, what can you say about a satisfying assignment for $\mathcal{C}$? Explain.

8. For some `2SAT` instance $\mathcal{C}$, is it possible to know with certainty whether or not $\mathcal{C}$ is satisfiable by making exactly one query to a `Reachability` oracle and assuming no other knowledge about $\mathcal{C}$, including its size? Defend your answer.

9. Draw the implication graph for the following set of binary clauses.

$$(\overline{x}_2, \overline{x}_3), (x_2, \overline{x}_4), (x_1, \overline{x}_3), (x_2, x_3), (x_1, x_4), (\overline{x}_1, x_4), (x_1, \overline{x}_2).$$

Perform the Improved 2SAT Algorithm to determine a satisfying assignment for this set of clauses. Hint: remember that the first literal tested should be x_1, followed by $\overline{x}_1$ if necessary.

10. Repeat the previous problem, but now add the additional clause $(\overline{x}_2, x_3)$. Verify that there is now an inconsistent cycle in the implication graph by performing the `Improved 2SAT` algorithm and witnessing a variable x_i for which both R_{x_i} and $R_{\overline{x}_i}$ are inconsistent. Follow the hint from Exercise 9.

11. Draw the implication graph for the following instance of `2SAT`.

$$\mathcal{C} = \{(x_2, \overline{x}_4), (\overline{x}_2, x_5), (x_4, x_6), (\overline{x}_2, \overline{x}_4), (\overline{x}_5, \overline{x}_6), (\overline{x}_1, x_3), (x_1, \overline{x}_3), (x_3, \overline{x}_5)\}.$$

Perform the `Improved 2SAT` algorithm to determine a satisfying assignment for this set of clauses. Follow the hint described in Exercise 9. Another hint: the final satisfying assignment should equal the union of two different consistent reachability sets.

12. In the `2SAT` algorithm, suppose the oracle answers `yes` to `reachable`$(G_{\mathcal{C}}, \overline{x}_3, x_3)$, but `no` to `reachable`$(G_{\mathcal{C}}, x_3, \overline{x}_3)$. Then if $\mathcal{C}$ is a unique satisfying assignment α, then what can you say about α?

Additional Exercises

A. For the `Reachability` algorithm, use math induction to prove that any vertex x that gets marked is reachable from u. Hint: assign an index to each marked vertex that represents the distance of that vertex from u. In particular, assign u index 0. Then if vertex w has assigned index i and (w, x) is the edge responsible for the marking of x, then assign x index $i + 1$. Perform the induction on the index i assigned to vertex x.

B. For the directed graph $G = (V, E)$, where

$$V = \{a, b, c, d, e, f, g, h, i, j, k\}$$

and the edges are given by

$$E = \{(a, b), (a, c), (b, c), (b, d), (b, e), (b, g), (c, g), (c, f),$$

$$(d, f), (f, g), (f, h), (g, h), (i, j), (i, k), (j, k)\},$$

use the Reachability Algorithm to determine if vertex k is reachable from vertex a. Show the contents of the FIFO queue Q at each stage of the algorithm.

Solutions to Core Exercises

1. a) is false since 128 is not a power of 4. b) is false since a member of $\mathcal{P}(\{a, b, c, d, e\})$ must be a subset of $\{a, b, c, d, e\}$. But c is a member of that set, not a subset of the set. c) is true. d) is true since, e.g., the next three numbers in the set are 25, 32, and 40, and so 37 is not a member of this set.

2. $f(1) = \emptyset$ since 1 is not prime. $f(28) = \{2, 7\}$, and $f(2025) = \{3, 5\}$.

3. We have

$$\mathcal{S} = \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}, \{10, 11, 12\}, \{13, 14, 15\}, \{1\}, \{2\}, \{3\}, \{4\}\}$$

   satisfies the requirements. The union of the first five sets equals $\{1, \ldots, 15\}$ but the union of any four of the subsets does not equal $\{1, \ldots, 15\}$.

4. Satisfying assignment: $\alpha = (x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 1)$.

5. $P = x_7, x_1, \overline{x}_5, x_3, \overline{x}_2$.

6. Using the fact that $P \to Q$ is logically equivalent to $\overline{P} \vee Q$, we have

$$\mathcal{C}' = \{(\overline{x}_3, \overline{x}_4), (x_4, x_5), (\overline{x}_5, x_1), (\overline{x}_1, x_4), (\overline{x}_4, \overline{x}_2), (x_2, x_3)\}.$$

   If $x_4 = 0$, then clauses

$$(x_4, x_5), (\overline{x}_5, x_1), (\overline{x}_1, x_4)$$

   cannot all be satisifed. Indeed, if $x_4 = 0$, then the first clause forces $x_5 = 1$, which forces the second clause to assign $x_1 = 1$, which forces the third clause to assign $x_4 = 1$, which is a contradiction. Similarly, if $x_4 = 1$, then clauses

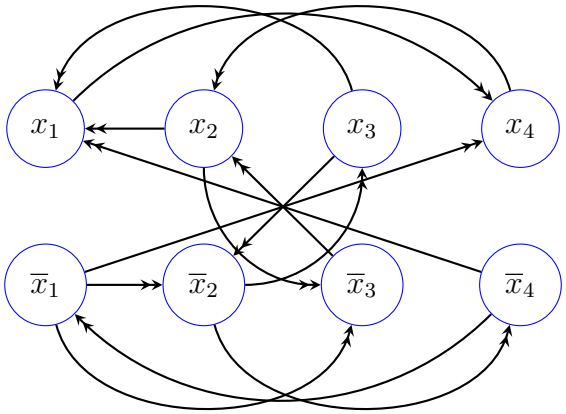
$$(\overline{x}_4, \overline{x}_2), (x_2, x_3), (\overline{x}_3, \overline{x}_4)$$

   cannot all be satisfied. Indeed, if $x_4 = 1$, then the first clause forces $x_2 = 0$, which forces the second clause to assign $x_3 = 1$, which forces the third clause to assign $x_4 = 0$, which is a contradiction. Thefore, any implication graph that contains such an inconsistent cycle as C leads to the original `2SAT` instance $\mathcal{C}$ being unsatisfiable.

7. The satisfying assignment must assign $x_3 = 0$, since, based on the query answer, there is a path from x_3 to $\overline{x}_3$ which means the assumption that $x_3 = 1$ leads to a contradiction.

8. No, two queries at a minimum are needed. For example, what is the most one can say if the query `reachable`$(G_{\mathcal{C}}, x_3, \overline{x}_3)$ were answered "yes"? "no"?

9. The implication graph $G_{\mathcal{C}}$ is shown below. The reachability set for $l = x_1$ is $R = \{x_1, x_2, \overline{x}_3, x_4\}$ and α_R satisfies $\mathcal{C}$



10. We have
$$R_{x_1} = R_{\overline{x_1}} = \{x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}\},$$
are both inconsistent. Therefore, $\mathcal{C}$ is unsatisfiable.

11. First recursive case: $R_{x_1} = \{x_1, x_3\}$. Second recursive case:
$$R_{x_2} = \{x_2, x_5, \overline{x}_6, x_4, \overline{x}_2, \overline{x}_4, x_6, \overline{x}_5\}$$
is inconsistent. However, $R_{\overline{x}_2} = \{\overline{x}_2, \overline{x}_4, x_6, \overline{x}_5\}$ is consistent. Satisfying assignment: $\alpha = (x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 1)$.

12. $\alpha(x_3) = 1$, since R_{x_3} is consistent and $R_{\overline{x}_3}$ is inconsistent. Therefore, no satisfying assignment can assign 0 to x_3.

Solutions to Additional Exercises

A. **Basis step.** Suppose x is marked and has index 0. Then necessarily $x = u$ and so x is reachable from u.

Inductive step. Assume that for some $i \geq 0$, any marked vertex w that has an assigned index of i is reachable from u. Show that any marked vertex with assigned index $i + 1$ is also reachable from u.

Proving the inductive step. Let x be a marked vertex that has been assigned index $i + 1$. Let (w, x) be the edge responsible for the marking of x. Then w has assigned index i and by the inductive assumption is reachable from u. But then x is also reachable from u via a path from u to w, followed by traversing the edge (w, x).

B. Queue sequence: $Q_1 = \{a\}$, $Q_2 = \{b, c\}$, $Q_3 = \{c, d, e, g\}$, $Q_4 = \{d, e, g, f\}$, $Q_5 = \{e, g, f\}$, $Q_6 = \{g, f\}$, $Q_7 = \{f, h\}$, $Q_8 = \{h\}$, $Q_9 = \emptyset$. Therefore, vertex k is not reachable from a since it was never marked and added to Q.