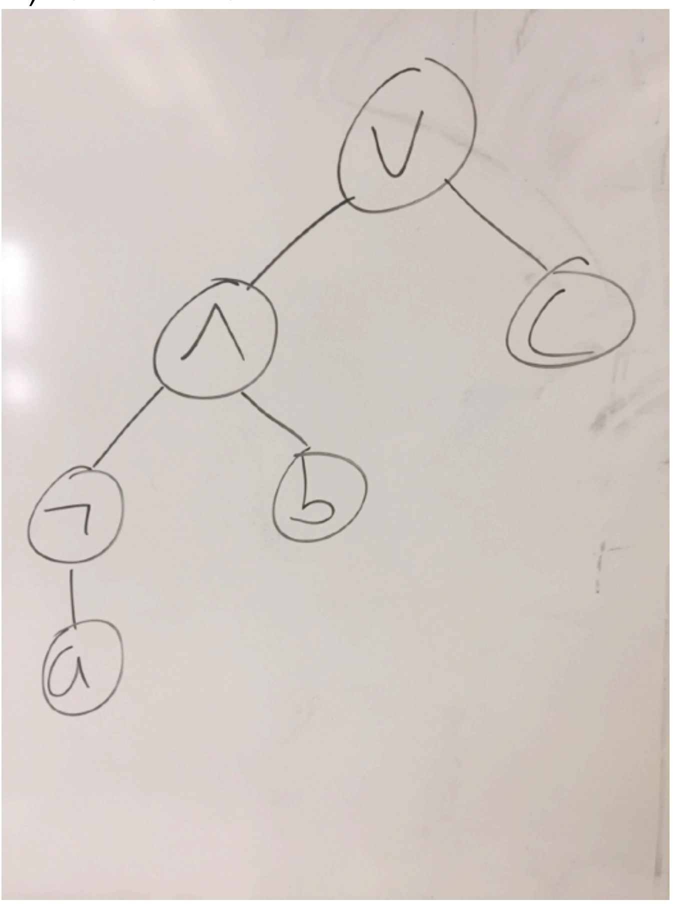
**COMP 410**
**Fall 2023**
**Midterm Practice Exam #1 (Solutions)**

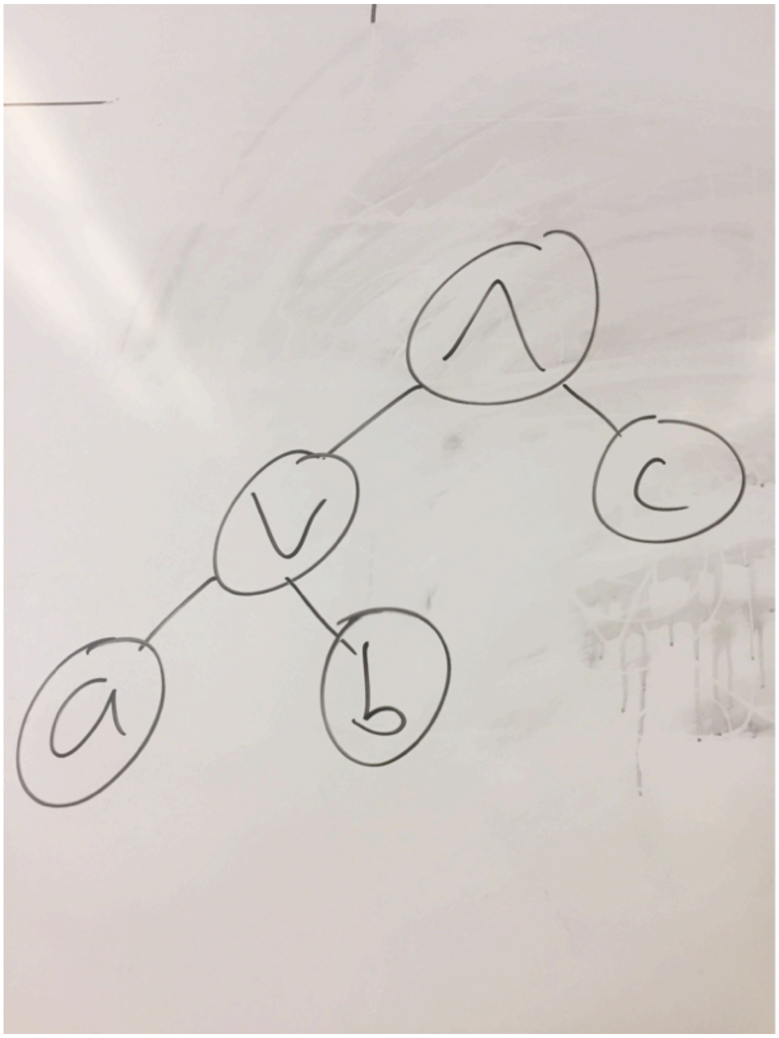
## Abstract Syntax Trees

In Boolean expressions, ¬ has the highest precedence, followed by ∧ and ∨. With this in mind, write out the ASTs corresponding to each of the following Boolean expressions:

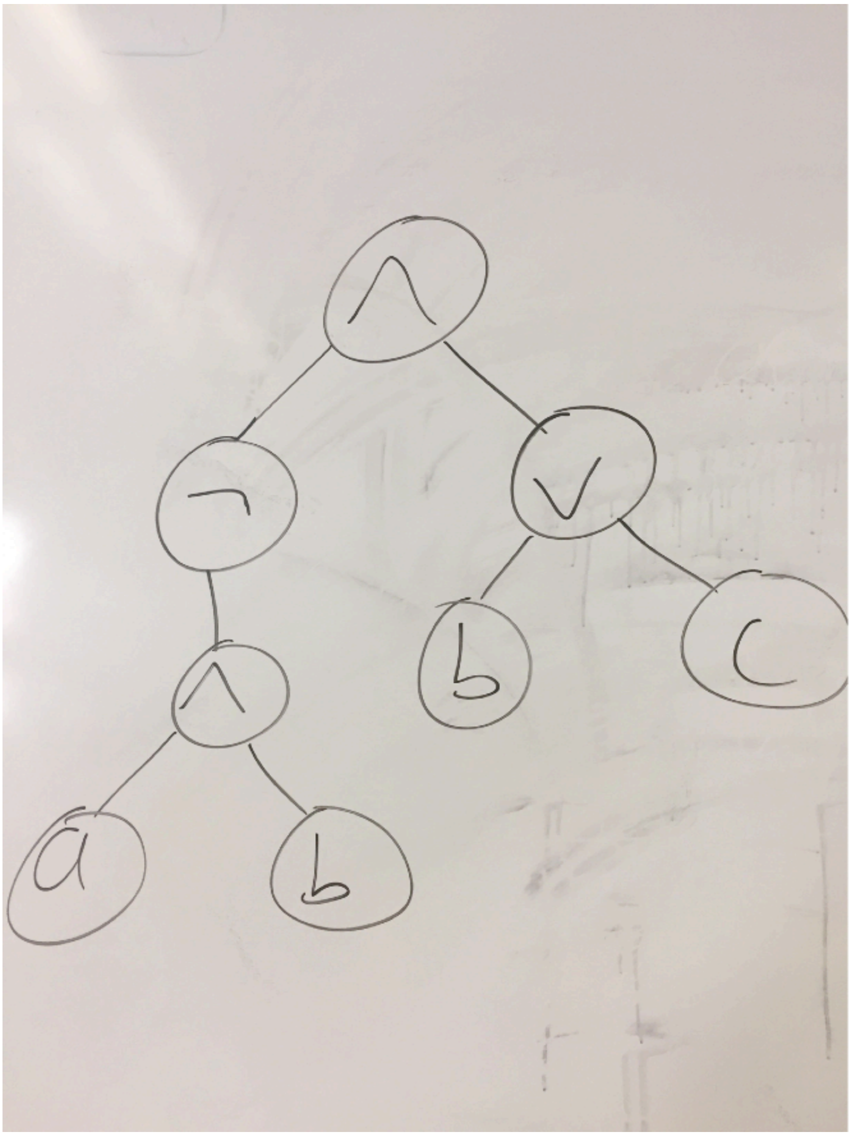
1.) `¬a ∧ b ∨ c`



2.) (a ∨ b) ∧ c

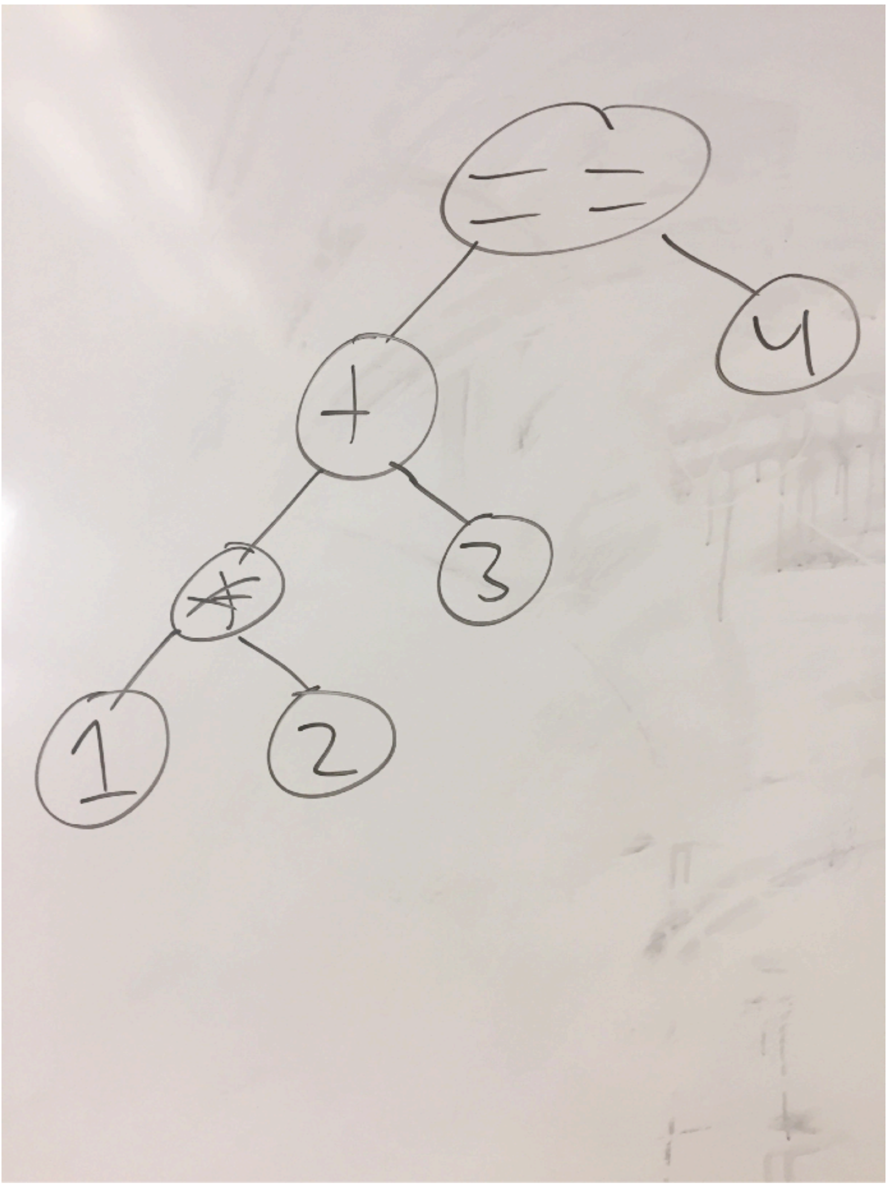


3.) $\neg(a \wedge b) \wedge (b \vee c)$

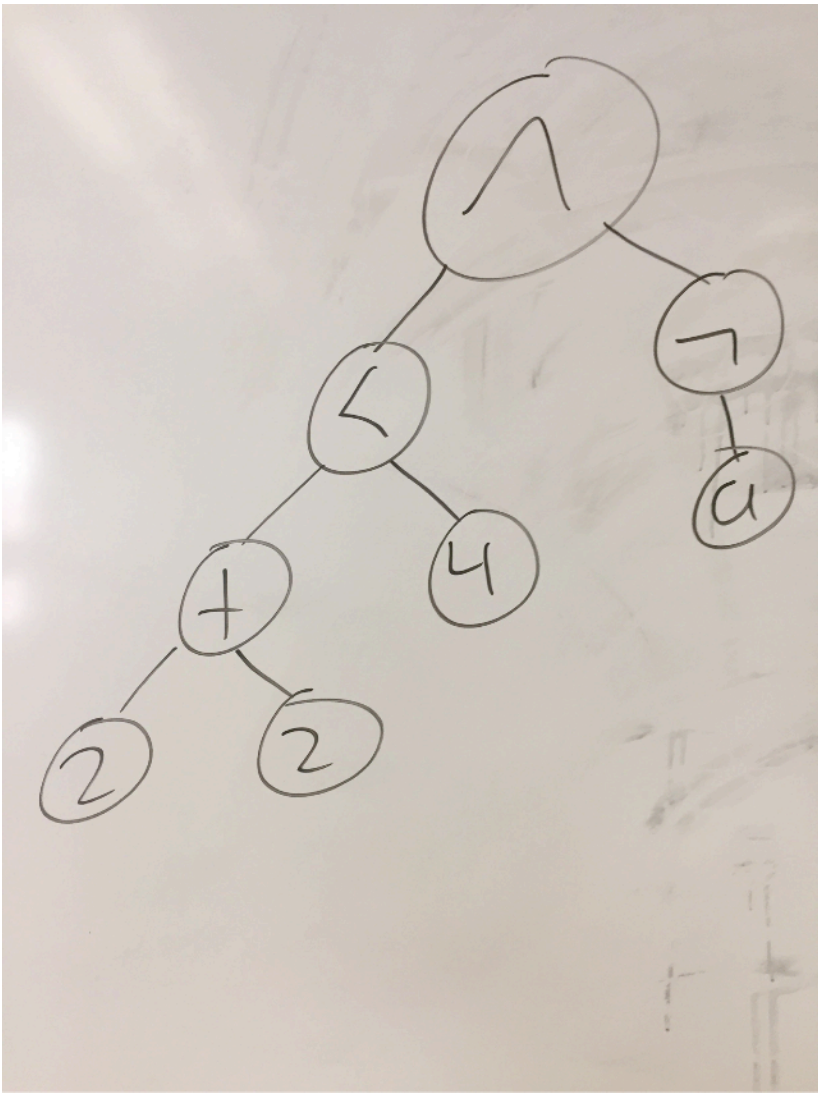


Arithmetic expressions can be used to form Boolean expressions with the help of arithmetic comparisons (e.g., <, <=, >, >=, ==). These comparisons have the lowest possible precedence. With this in mind, write out the ASTs corresponding to each of the following expressions:

4.) `1 * 2 + 3 == 4`



5.) (2 + 2 < 4) ∧ ¬a



6.) Consider the following Python class definitions, which are adapted from assignment 1's boolean evaluator. These classes are used to represent AST nodes.

```
class And:
  def __init__(self, left, right):
    self.left = left
    self.right = right

class Or:
  def __init__(self, left, right):
    self.left = left
    self.right = right
```

Assume that Boolean true is represented as an AST with Python's `True`, and Boolean false is represented as an AST with Python's `False`. With all this in mind, represent the following Boolean expressions in Python using `And`, `Or`, `True`, and `False` as appropriate.

6.a) `true ∧ false`

`And(True, False)`

6.b.) `false ∨ true`

`Or(False, True)`

6.c.) `false ∧ true ∨ true`

`Or(And(False, True), True)`

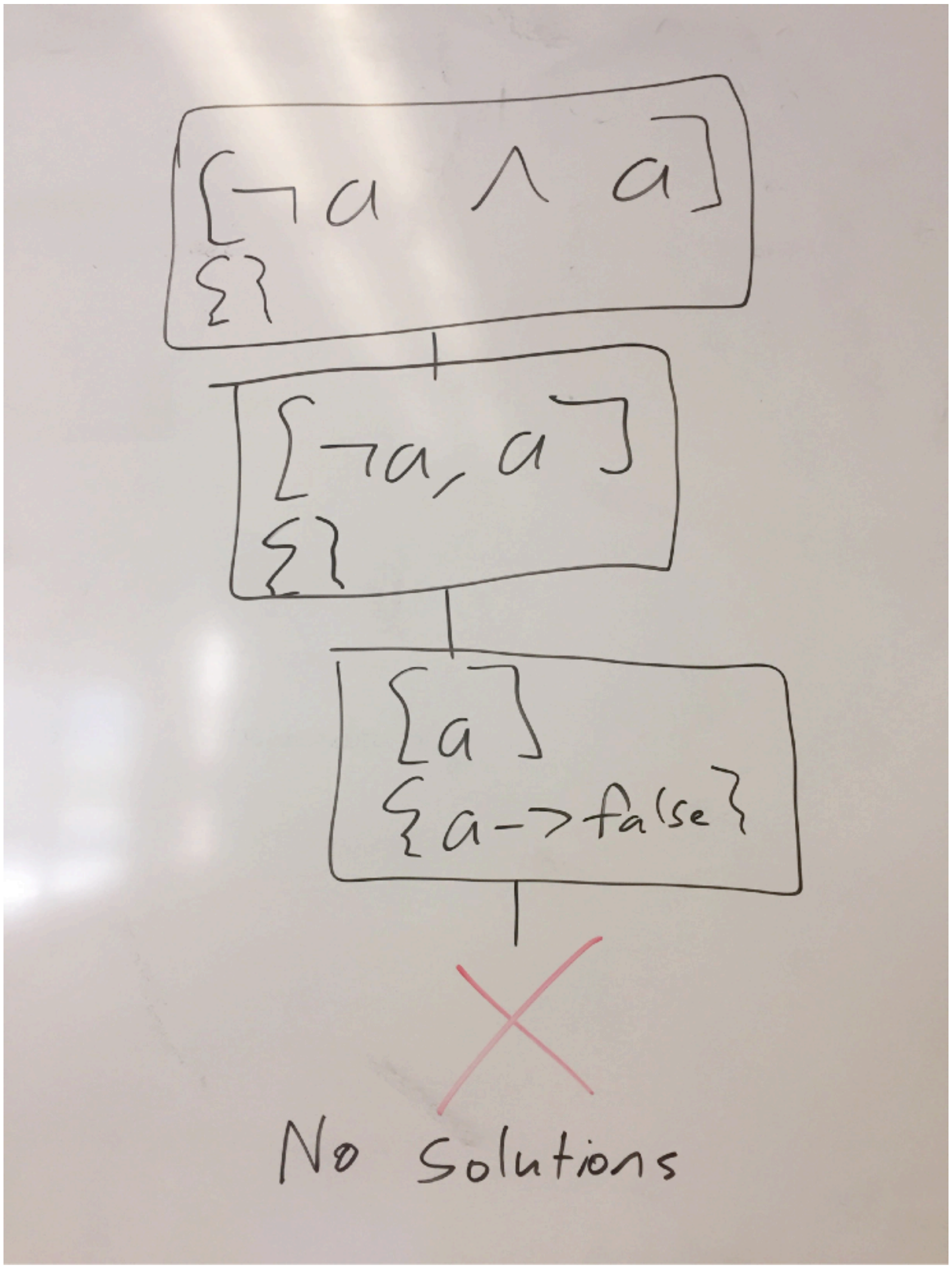
6.d.) `false ∨ true ∧ true`

`Or(False, And(True, True))`

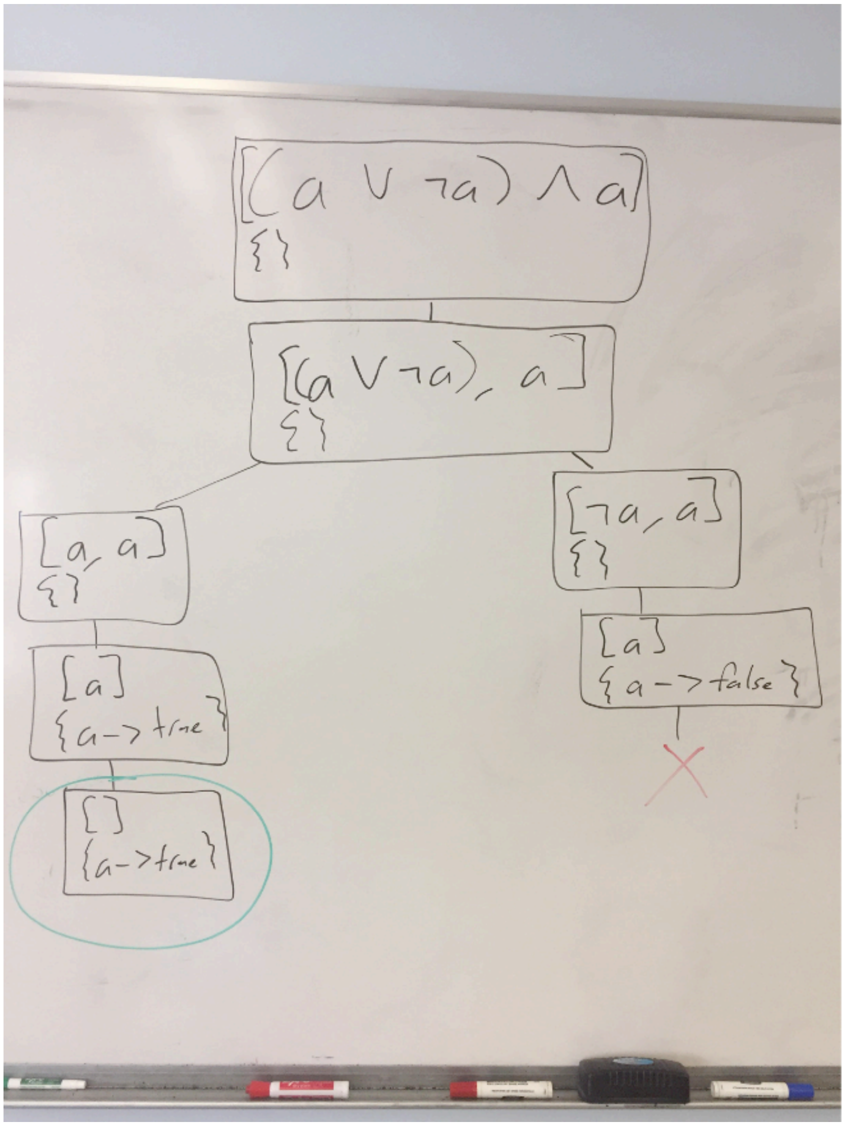
**Semantic Tableau**

For each of the following Boolean formulas, write out the complete semantic tableau tree. **Circle** the nodes in the tree representing solutions. If a tree has no solutions, say so. **Be sure to write all steps.**

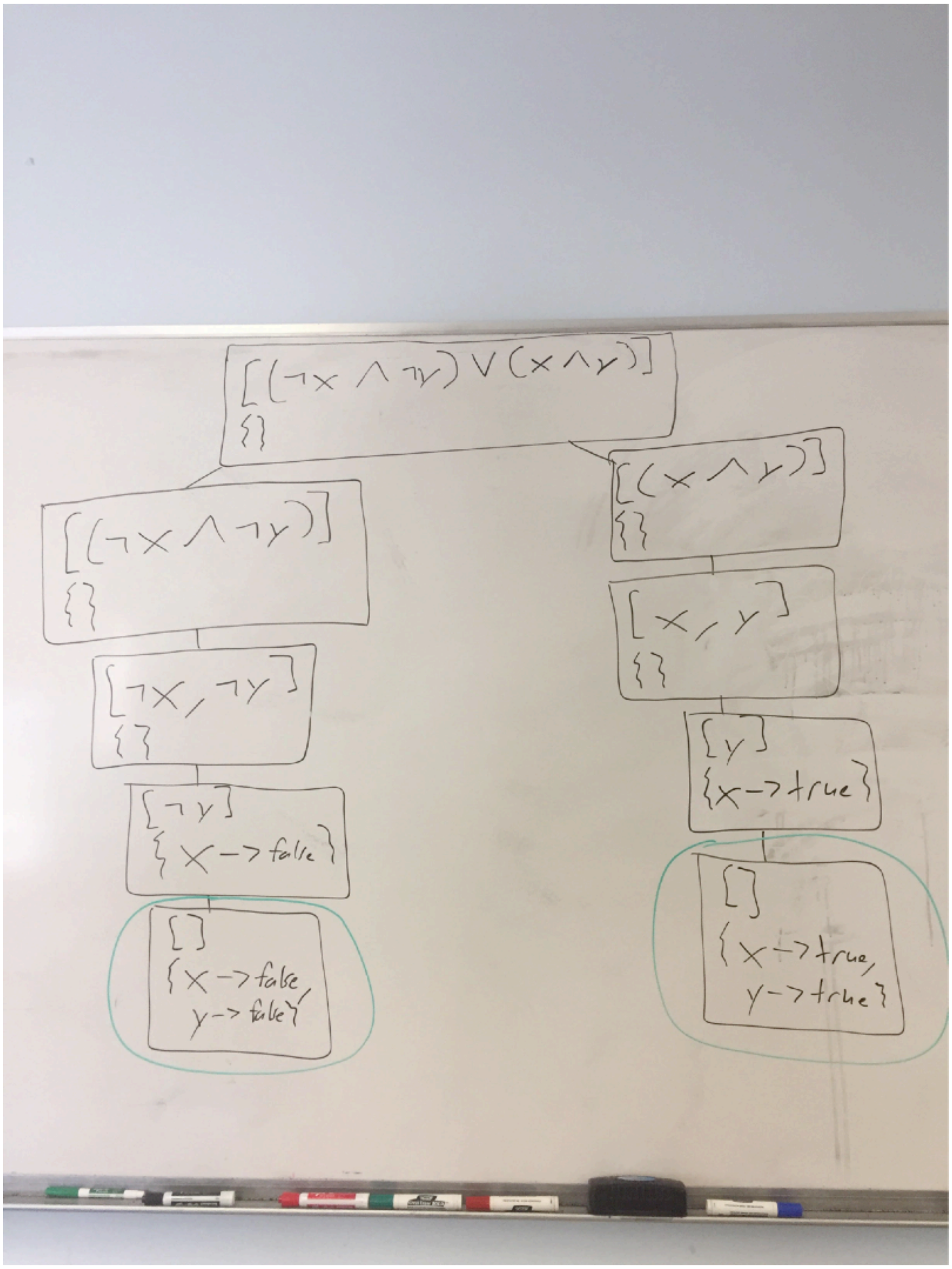
7.) ¬a ∧ a



8.) (a ∨ ¬a) ∧ a



9.) (¬x ∧ ¬y) ∨ (x ∧ y)



**Prolog - Modeling the World**

10.a)
For this problem, you need to write a clause database encapsulating pricing information for a convenience store. Write Prolog code accurately reflecting the following:

- Soda costs $2
- Chips cost $3
- Hot dogs cost twice as much as soda (do not hardcode $4)
- Soda chips, and hot dogs are food
- Pencils and pens are office supplies
- All office supplies cost $2
- Cold medicine costs $7

```
% all facts and rules with the same name should be placed
% together in the file
cost(soda, 2).
cost(chips, 3).
cost(hot_dog, Cost) :-
    cost(soda, SodaCost),
    Cost is SodaCost * 2.
cost(OS, 2) :-
    office_supplies(OS).
cost(cold_medicine, 7).

food(soda).
food(chips).
food(hot_dog).

office_supplies(pencil).
office_supplies(pen).
```

Using the clause database you previously wrote, write queries to determine the following:

10.b.) Which items cost exactly $2?

```
?- cost(Item, 2).
```

10.c.) Which items cost more than $3?

```
?- cost(Item, Cost), Cost > 3.
```

10.d.) Which foods cost less than $3?

```
?- food(Food), cost(Food, Cost), Cost < 3.
```

10.e.) Which foods are also office supplies?

```
?- food(Item), office_supplies(Item).
```

**Unification**

Consider each of the following unification attempts. If the unification succeeds, record any values any variables take. If the unification fails, say so.

11.) `foo(1, X) = foo(Y, 2)`

```
X = 2, Y = 1
```

12.) `foo(1, X) = foo(X, 2)`

`false`

13.) `foo(1, _) = foo(X, 2)`

`X = 1`

14.) `foo(1, _) = foo(1, _)`

`true`

15.) `foo(1, 2, bar) = foo(X, _, _, _)`

`false`

16.) `foo(bar(baz), X) = foo(Y, Z), Y = bar(A)`

`X = Z, Y = bar(baz), A = baz`

## Recursion

17.) Consider the following mathematical definition of a recursive function:

$$f_n = \begin{cases} 2 & \text{if } n = 0 \\ 3 & \text{if } n = 1 \\ (3 \times f_{n-1}) + (4 \times f_{n-2}) & \text{otherwise} \end{cases}$$

Write an equivalent definition in Prolog.

```
f(0, 2).
f(1, 3).
f(N, Result) :-
    N > 1,
    MinOne is N - 1,
    MinTwo is N - 2,
    f(MinOne, T1),
    f(MinTwo, T2),
    Result is (3 * T1) + (4 * T2).
```

18.) Write a procedure named `evensBetween`, which will nondeterministically produce all the even numbers within an inclusive range. As a hint, a number `N` is even if and only if the clause `0 is mod(N, 2)` is true. An example query is below:

```
?- evensBetween(1, 4, Even).
Even = 2 ;
Even = 4.
```

```
evensBetween(Min, Max, Min) :-
    Min   =< Max,
    0 is mod(Min, 2).
evensBetween(Min, Max, Result) :-
    Min   < Max,
    NewMin is Min + 1,
    evensBetween(NewMin, Max, Result).
```