Daily Practice Problem (DPP)

Topic: Satisfiability Problem - Sudoku, Rules of Inference

Name: _____

1. Show that the negation of an unsatisfiable compound proposition is a tautology and the negation of a compound proposition that is a tautology is unsatisfiable.

2. Determine whether each of these compound propositions is satisfiable.

   (a) $(p \vee \neg q) \wedge (\neg p \vee q) \wedge (\neg p \vee \neg q)$

   (b) $(p \to q) \wedge (p \to \neg q) \wedge (\neg p \to q) \wedge (\neg p \to \neg q)$

   (c) $(p \leftrightarrow q) \wedge (\neg p \leftrightarrow q)$

3. Determine whether each of these compound propositions is satisfiable.

   (a) $(p \vee q \vee \neg r) \wedge (p \vee \neg q \vee \neg s) \wedge (p \vee \neg r \vee \neg s) \wedge (\neg p \vee \neg q \vee \neg s) \wedge (p \vee q \vee \neg s)$

   (b) $(\neg p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg s) \wedge (p \vee \neg q \vee \neg s) \wedge (\neg p \vee \neg r \vee \neg s) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg r \vee \neg s)$

   (c) $(p \vee q \vee r) \wedge (p \vee \neg q \vee \neg s) \wedge (q \vee \neg r \vee s) \wedge (\neg p \vee r \vee s) \wedge (\neg p \vee q \vee \neg s) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee \neg q \vee s) \wedge (\neg p \vee \neg r \vee \neg s)$

4. Explain how a 4×4 SUDOKU puzzle can be modeled as a SATISFIABILITY PROBLEM. Mention the types of logical constraints needed.

5. Construct a compound proposition that asserts that every cell of a 9×9 SUDOKU puzzle contains at least one number.

6. Let $P(x)$: "Cell x contains number 1" for a 4×4 Sudoku grid. Write the proposition for:

   "Every row contains the number 1 exactly once."

7. Explain the steps in the construction of the compound proposition given in the text that asserts that each of the nine 3×3 blocks of a 9×9 Sudoku puzzle contains every number.

Connecting Logic With Computer Science

8. Search engines allow users to specify Boolean conditions in their queries. For example, "social OR networks" will return only web pages containing either the word "social" or the word "networks." You can view a query as a predicate Q; the search engine returns (in some order) the list of pages p for which $Q(p)$ is true. Consider the following queries:

    Query A: "*java AND program AND NOT computer*"
    Query B: "*(computer OR algorithm) AND java*"
    Query C: "*java AND NOT (computer OR algorithm OR program)*"

    Give an example of a web page—or a sentence—that would be returned as specified in these problems:

    a. Returned by query A but not by B or C.

    b. Returned by query B but not by A or C.

    c. Returned by query C but not by A or B.

    d. Returned by query A and B but not by C.

9. Using the atomic propositions given below, translate the following statements about Python expressions into logical notation:

p:	`x + y` is valid Python	u:	`x` is a numeric value
q:	`x * y` is valid Python	v:	`y` is a numeric value
r:	`x ** y` is valid Python	w:	`x` is a list
s:	`x * y` is a list	z:	`y` is a list
t:	`x + y` is a list		

(Exercises (a-f) make accurate statements about expressions involving numbers and lists in Python, but even so they do not come close to fully characterizing the set of valid Python statements, for multiple reasons: first, they're about particular variables—`x` and `y`—rather than about generic variables. And, second, they omit some common-sense facts, like the fact that it's not simultaneously possible to be both a list and a numeric value.)

(a) $x * *y$ is valid Python if and only if x and y are both numeric values.

(b) $x + y$ is valid Python if and only if x and y are both numeric values, or they're both lists.

(c) $x * y$ is valid Python if and only if x and y are both numeric values, or if one of x and y is a list and the other is numeric.

(d) $x * y$ is a list if $x * y$ is valid Python and x and y are not both numeric values.

(e) if $x + y$ is a list, then $x * y$ is not a list.

(f) $x + y$ and $x * *y$ are both valid Python only if x is not a list.

10. In programming, an assertion is a logical statement that announces ("asserts") a condition that the programmer believes to be true. For example, a programmer who is about to access the 202nd element of an array A might assert that length(A) 202 before accessing this element. When an executing program reaches an assert statement, the program aborts if the condition in the statement isn't true. (Using assertions can be an extremely valuable way of documenting and debugging programs, particularly because liberally including assertions will allow the revelation of unexpected data values much earlier in the execution of a program. And these languages have a global toggle that allows the testing of assertions to be turned off, so once the programmer is satisfied that the program is working properly, she doesn't have to worry about any running-time overhead for these checks.)

    (a) Give a nonempty input array A[1 . . . n] that would cause the assertion in Figure–a to fail. (That is, identify an array A that would cause for the asserted condition to be false.)

    (b) Give a nonempty input array A[1 . . . n] that would cause the assertion in Figure–b to fail.

    (c) Give a nonempty input array A[1 . . . n] that would cause the assertion in Figure–c to fail.

(a)

```
1 last := 0
2 for index := 1, ..., n − 1:
3     if A[index] > A[index+1] then
4         last := index
5 assert last ≥ 1 and last ≤ n − 1
6 swap A[last] and A[last + 1]
```

(b)

```
1 total := A[1]
2 i := 1
3 for i := 2, ..., n − 1:
4     if A[i + 1] > A[i] then
5         total := total + A[i]
6         assert total > A[1]
7 return total
```

(c)

```
1 for start := 1, ..., n − 1:
2     min := start
3     for i := start + 1, ..., n:
4         assert start = 1 or A[i] > A[start − 1]
5         if A[min] > A[i] then
6             min := i
7         swap A[start] and A[min]
```

Challenge Problem

(a)

```
1 for x in S:
2     for y in S:
3         flag := False
4         if P(x) or P(y) then
5             flag := True
6     if flag then
7         return True
8 return False
```

(b)

```
1 for x in S:
2     flag := False
3     for y in S:
4         if not P(x, y) then
5             flag := True
6     if flag then
7         return True
8 return False
```

(c)

```
1 for x in S:
2     flag := True
3     for y in S:
4         if not P(x, y) then
5             flag := False
6     if flag then
7         return True
8 return False
```

(d)

```
1 for x in S:
2     flag := False
3     for y in S:
4         if not P(x, y) then
5             flag := True
6     if not flag then
7         return False
8 return True
```

(e)

```
1 for x in S:
2     for y in S:
3         if P(x, y) then
4             return False
5 return True
```

(f)

```
1 flag := False
2 for x in S:
3     for y in S:
4         if P(x, y) then
5             flag := True
6 return flag
```

Figure Some nested loops that compute some facts about a predicate P. (Assume that the universe S is finite.)

(a) The code in Figure a uses nested loops to compute some fact about a predicate P. Write a fully quantified statement of predicate logic whose truth value matches the value returned by the code. (Assume that S is a finite universe.)

(b) Do the same for the code in Figure b.

(c) Do the same for the code in Figure c.

(d) Do the same for the code in Figure d.

(e) Do the same for the code in Figure e.

(f) Do the same for the code in Figure f.

Self-Evaluation

How confident do you feel after solving this practice sheet?

Low 1 2 3 4 5 High