CS 4100 Homework 01

Due Thursday 1/19 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

You must submit the homework as an `.ipynb` file in Gradescope by the due date and time. All homeworks will be scored with a maximum of 100 points; if point values are not given for individual problems, then all problems will be counted equally.

Please select `Kernel -> Restart and Run All` before you submit; we will not run your code while grading, and anything you do not demonstrate as correct will be assumed to be incorrect.

The goals of this first homework are that you

1. Get up to speed on Python and Jupyter Notebooks (by going through the various tutorials or other resources as needed);
2. Practice the submission process through Gradescope (and allow us to practice the grading process); and
3. Get started on the material in Chapter 2 of the textbook.

Whenever possible, you will have all the material you need for homeworks the week before they are due (so that you have the weekend before the due date to complete it). However this will not always be possible, and in this homework, I am giving you some exercises on material we will cover on Monday 1/16; these should be very familiar from your Discrete Mathematics course in any case.

Problem 1

Sign up for an account for `chatGPT` :

https://openai.com/blog/chatgpt/

and ask it for advice on how to prepare for your Introduction to AI class. If the system is busy, you may have to try it late at night or early in the morning.

Take a screen shot of your interaction. In the solution cell following, drag and drop your screen shot file; when you run the cell, the image will display.

Note: Sometimes when I do the drag and drop, nothing happens; I think this is when you try to drop an image file into a cell more than once, but I'm not precisely sure of the problem. If you simply create a new cell and try again, it should work.

Solution 1

WA How should I prepare for my class in Introduction to AI?

To prepare for an introduction to AI class, you should have a solid understanding of basic programming concepts and mathematical concepts such as probability and statistics. Familiarity with calculus and linear algebra is also helpful. Additionally, you should be comfortable with problem-solving and critical thinking.

You may also want to read up on the history and current state of AI research, as well as familiarize yourself with the different subfields within AI, such as machine learning, computer vision, and natural language processing.

Finally, it's important to stay current with the latest news and developments in the field by following relevant research journals and publications. And if the class allow it, you can also take some online course or tutorials to get more familiar with the field.

Problem 2

For this problem, we will explore chatGPT's coding ability.

Note: To input multi-line prompts, at the end of the line use Shift-Return instead of Return and it will not submit the prompt.

Part 2.1

Ask `chatGPT` whether the following code will correctly find the larger of the two integers x and y:

```
if(x > y):
   print(x)
```

and observe the result. Then ask if the following code will correctly print the largest element in a list L:

```
m = -1
for x in L:
   if(x > m):
       m = x
print(m)
```

and observe the result.

(Here is what happens:

WA

Does the following code correctly print the larger of the two integers x and y?

if(x>y):
  print(x)a

No, the code you provided does not correctly print the larger of the two integers x and y.

The code uses an if statement to check if the value of x is greater than y, and if it is, then it will print the value of x. However, it does not handle the case where x and y are equal, and also it does not handle the scenario when x is less than y.

To handle the scenario when x and y are equal, you can add an additional check `elif x == y:` and then print x or y

Copy code

```
if (x > y):
  print(x)
elif x == y:
    print(x)
else:
    print(y)
```

WA

Does the following code correctly print the largest integer in the list L?

```
m = -1
for x in L:
  if(x > m):
    m = x
print(m)
```

Yes, the code you provided correctly finds and prints the largest integer in the list L.

It assigns the value of -1 to the variable m, then it iterates through the list L using a for loop. For each element x in the list, it compares it to the current maximum value stored in m. If the current element x is greater than m, the value of m is updated to the value of the current element x. Finally, the largest integer in the list is printed by the last line `print(m)`.

It is a good practice to initialize m with a low value, such as -infinity, when you are looking for the maximum element, this way you will be sure that the first element in the list will be stored as max.

It is also important to note that this code will work only if the list L contains integers, if it contains other type of data, you will get a TypeError.

Ok, your turn! Play around with various examples and find a relatively complex program that `chatGPT` appears to understand correctly. You may want to ask something more than just "is this correct?"

As with Problem 1, paste your screenshot in the solution cell.

Solution 2.1

(Any answer is fine.)

Part 2.2

Now play around with various examples and find a relatively simple program that `chatGPT` does NOT understand correctly. Paste your screenshot in the solution cell.

Solution 2.2

(Any answer is fine.)

Part 2.3

Ask `chatGPT` to write a Python program to determine if an integer n is a prime. Observe the result.

(Here is what happens.)

WA

Please write code in Python that checks whether a number n is prime.

Here's an example of a Python function that checks whether a number n is prime:

Copy code

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
```

This function first checks if the number is less than 2, in which case it returns False as 0 and 1 are not prime numbers. Then it uses a for loop to check if the number is divisible by any integers from 2 to the square root of n. If the number is divisible by any of these integers, the function returns False as it's not prime. If the for loop completes without finding any divisors, the function returns True, indicating that the number is prime.

Play around with various examples where `chatGPT` writes Python code and give an interesting example where it creates a reasonably complex program. Try to add details or constraints so that it is clearly not just spitting out something from StackOverflow.

Solution 2.3

(Any answer is fine.)

Part 2.4 (Not exactly about coding, but similar)

Ask `chatGPT` to solve exercise 2.1 from the textbook and display the result as usual in the solution cell. Also evaluate whether it did the problem correctly.

Solution 2.4

WA

Please give a Backus-Naur form grammer for formulae in propositional logic

A Backus-Naur Form (BNF) grammar for formulae in propositional logic can be expressed as follows:

Copy code

```
<formula> ::= <atomic_formula> | <negation> | <binary_formula>

<atomic_formula> ::= <prop_constant> | <prop_variable>

<prop_constant> ::= 'T' | 'F'

<prop_variable> ::= 'p' | 'q' | 'r' | ...

<negation> ::= '~' <formula>

<binary_formula> ::= '(' <formula> <binary_connective> <formula> ')'

<binary_connective> ::= '&' | '|' | '->' | '<->'
```

This grammar defines a formula as either an atomic formula (a propositional constant or variable), a negation of a formula, or a binary formula (i.e. a formula composed of two subformulas connected by a binary connective). The binary connectives can be '&' (and), '|' (or), '->' (implication) and '<->'(equivalence) .

This seems correct. However, that is not surprising, since it probably just found it on StackOverflow or some Wikipedia page!

Part 2.5

How well do you think `chatGPT` would work as a coding assistant? Give a brief assessment with pros and cons about the practical use of the current system based on your own coding experiences.

Solution 2.5

Any thoughtful answer is fine. You must discuss pros and cons as required.

Problem 3

Do problem 2.2 c & d and provide the solutions using Markdown and Latex, or by writing it on paper, scanning it, and dragging and dropping it into the solution cell.

Solution 3

(c)

A	B	(A→B)	(B→A)	((A→B)∧(B→A)) (α)	(A⇔B) (β)	α⇔β
F	F	T	T	T	T	T
F	T	T	F	F	F	T
T	F	F	T	F	F	T
T	T	T	T	T	T	T

(d)

A	B	C	(A∨B)	(¬B∨C)	((A∨B)∧(¬B∨C)) (α)	(A∨C) (β)	α→β
F	F	F	F	T	F	F	T
F	F	T	F	T	F	T	T
F	T	F	T	F	F	F	T
F	T	T	T	T	T	T	T
T	F	F	T	T	T	T	T
T	F	T	T	T	T	T	T
T	T	F	T	F	F	T	T
T	T	T	T	T	T	T	T

Problem 4

Do problem 2.3 b & c and provide the solutions using Latex in the solution cell.

Solution 4

(b)

$$\begin{aligned} A \wedge B \iff A \vee B &= (A \wedge B \implies A \vee B) \wedge (A \vee B \implies A \wedge B) \\ &= (\neg(A \wedge B) \vee A \vee B)) \wedge (\neg(A \vee B) \vee (A \wedge B)) \\ &= (\neg A \vee \neg B \vee A \vee B) \wedge ((\neg A \wedge \neg B) \vee (A \wedge B)) \\ &= (\neg A \vee \neg B \vee A \vee B) \wedge ((\neg A \wedge \neg B) \vee A) \wedge ((\neg A \wedge \neg B) \vee B)) \\ &= (\neg A \vee \neg B \vee A \vee B) \wedge (\neg A \vee A) \wedge (\neg B \vee A) \wedge (\neg A \vee B) \wedge (\neg B \vee B) \end{aligned}$$

(d)

$$\begin{aligned} A \wedge (A \implies B) \implies B &= A \wedge (\neg A \vee B) \implies B \\ &= \neg(A \wedge (\neg A \vee B)) \vee B \\ &= (\neg A \vee (A \wedge \neg B)) \vee B \\ &= ((\neg A \vee A) \wedge (\neg A \vee \neg B)) \vee B \\ &= (\neg A \vee A \vee B) \wedge (\neg A \vee \neg B \vee B) \end{aligned}$$

Problem 5

Do problem 2.5 in Python and demonstrate your program on the three formulae in problem 2.4. As part of this process, you will need to convert these three formulae to conjunctive normal form.

Hint: It is a little messy to represent literals as strings, so a simple encoding could be to consider $\Sigma = \{A_1, A_2, A_3, \ldots\}$ and represent literals using just the subscript, with a negative literal being -1 * subscript. Thus, if your propositions are `gas_in_tank` and `car_starts` you could use the encodings:

1 = gas_in_tank
2 = car_starts
-1 = ¬ gas_in_tank
-2 = ¬ car_starts

The simplest representation of an interpretation/model in Python is simply a list of Boolean values, e.g.,

$$[True, False, False, True] \quad = \quad \{A_1 \mapsto t, A_2 \mapsto f, A_3 \mapsto f, A_4 \mapsto t\}.$$

Feel free to use another encoding if you have a better one!

In [1]:

```
# Solution 5

def satLiteral(I,L):
    i = abs(L)-1             # must - 1 because symbol indices start at 1, interpretation indices at 0
    if(L<0):                 # negative literal
        return not(I[i])
    else:
        return I[i]

def satClause(I,C):
    return any( [ satLiteral(I,L) for L in C ] )

def satCNF(I,A):
    return all( [ satClause(I,C) for C in A ] )

# Note that this can be done all in one (long) line! Interesting,
# but not particularly readable.

def satCNFOneLine(I,A):
    return all( [ any( [ (not(I[-L - 1]) if L < 0 else I[L - 1]) for L in C ] ) for C in A ] )
```

In [2]:

```
# test

'''
ex1

A
not B

ex2

A2 <- A4
A1 <- A2,A3
'''
ex1 = [ [1], [-2] ]

ex2 = [ [2,-4], [1,-2,-3] ]

'''
I1
A1
A1 = t A2 = f

I2
A1 = t A2 = t A3 = t A4 = t
'''

I1 = [True,False]
I2 = [ True, True, True, True ]

satCNF(I2,ex2)
```

Out[2]: True

In [3]:
```
from itertools import product

# utility to count how many symbols are needed

def numSymbols(A):
    return max( [abs(L) for C in A for L in C] )

def genInterpretations(N):
    return list(product([True,False], repeat=N))

def countInterpretations(A):
    return ( [ satCNF(I,A) for  I in genInterpretations(numSymbols(A)) ] ).count(True)
```

In [4]:
```
# test
ex = ex2  # ex1

for I in genInterpretations(numSymbols(ex)):
    print(I,"\t",satCNF(I,ex))

print("\n# of satisfying interpretations =",countInterpretations(ex))
```

```
(True, True, True, True)         True
(True, True, True, False)        True
(True, True, False, True)        True
(True, True, False, False)       True
(True, False, True, True)        False
(True, False, True, False)       True
(True, False, False, True)       False
(True, False, False, False)      True
(False, True, True, True)        False
(False, True, True, False)       False
(False, True, False, True)       True
(False, True, False, False)      True
(False, False, True, True)       False
(False, False, True, False)      True
(False, False, False, True)      False
(False, False, False, False)     True

# of satisfying interpretations = 10
```

In [5]:
```
def printSolution(A):
    n = countInterpretations(A)
    if(n == 0):
        print("The CNF formula is unsatisfiable.")
    elif(n == 2**numSymbols(A)):
        print("The CNF formula is true (valid).")
    else:
        print("The CNF formula is satisfiable.")
    print("There are", n, "satisfying interpretations.")
```

In [6]:
```
# Example 2.4.a

'''
(play_lottery and six_right) implies winner

    A1 = play_lottery
    A2 = six_right
    A3 = winner

       (1 & 2) -> 3
    => ~ (1 & 2) | 3
    => (~ 1 | ~ 2) | 3
    => (~ 1) | (~ 2) | 3

    A = [ [ -1, -2, 3 ] ]

'''
```

```
A = [ [ -1, -2, 3 ] ]

print("Answer for exercise 2.4.a:\n")

# for I in genInterpretations(numSymbols(A)):
#     print(I,"\t",satCNF(I,A))

printSolution(A)
```

```
Answer for exercise 2.4.a:

The CNF formula is satisfiable.
There are 7 satisfying interpretations.
```

In [7]:

```
# Example 2.4.b

'''
(play_lottery and six_right and (six_right implies win)) implies win

    A1 = play_lottery
    A2 = six_right
    A3 = winn

       (1 & 2 & (2 -> 3)) -> 3
    => ~(1 & 2 & (2 -> 3)) | 3
    => ~(1 & 2 & (~2 | 3)) | 3
    => (~1 | ~2 | (2 & ~3)) | 3
    => (~1 | ~2 | 3 | (2 & ~3))
    => (~1 | ~2 | ( ( 3 | 2 ) & (3 | ~3) )
    => (~1 | ( ( ~2 | 3 | 2 ) & ( ~2 | 3 | ~3 ) )
    => ( ( ~1 | ~2 | 3 | 2 ) & ( ~1 | ~2 | 3 | ~3 ) )

    B = [ [ -1, -2, 3, 2 ], [ -1, -2, 3, -3 ] ]

'''

B = [ [ -1, -2, 3, 2 ], [ -1, -2, 3, -3 ] ]

print("Answer for exercise 2.4.b:\n")

# for I in genInterpretations(numSymbols(B)):
#     print(I,"\t",satCNF(I,B))

printSolution(B)
```

```
Answer for exercise 2.4.b:

The CNF formula is true (valid).
There are 8 satisfying interpretations.
```

In [8]:

```
# Example 2.4.c

'''

    A1 = gas_in_tank
    A2 = car_starts

       ~( ( ~1 & (1 | ~2) ) -> ~2 )
    => ~( ~( ~1 & (1 | ~2) ) | ~2 )
    => ( ( ~1 & (1 | ~2) ) & 2 )
    => ( ~1 & (1 | ~2) ) & 2 )
'''

C = [ [ -1 ], [ 1, -2 ], [ 2 ] ]

print("Answer for exercise 2.4.c:\n")

# for I in genInterpretations(numSymbols(C)):
#     print(I,"\t",satCNF(I,C))

printSolution(C)
```

```
Answer for exercise 2.4.c:
```

```
The CNF formula is unsatisfiable.
There are 0 satisfying interpretations.
```