CS 560: Homework 1

Eric Stevens

October 7, 2018

Critique is in blue boxes:

While there are many difference between my answers and the provided solutions, I would argue that the differences are not substantive. Due to the dense nature of material, I rely heavily on the textbook for learning the material and therefore my answers are styled as I interpret the resources in the textbook. There appear to be several stylistic differences between the way material appears in the book and the way it is presented in class. I ask the grader to look past these stylistic differences and try to see whether I have the intuition about the material that the assignment requires. This critique will provide reasons why I believe I should not be heavily marked down for the differences between my solutions and the answer sheet.

Question 1

Part 1:

- **Interpretation 0**

$$\pi(a) = FALSE$$
$$\pi(b) = FALSE$$
$$\pi(c) = FALSE$$
$$\pi(d) = FALSE$$

- **Interpretation 1**

$$\pi(a) = TRUE$$
$$\pi(b) = FALSE$$
$$\pi(c) = FALSE$$
$$\pi(d) = FALSE$$

- **Interpretation 2**

$$\pi(a) = FALSE$$
$$\pi(b) = TRUE$$
$$\pi(c) = FALSE$$
$$\pi(d) = FALSE$$

- **Interpretation 3**

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = FALSE$$
$$\pi(d) = FALSE$$

- **Interpretation 4**

$$\pi(a) = FALSE$$
$$\pi(b) = FALSE$$
$$\pi(c) = TRUE$$
$$\pi(d) = FALSE$$

- **Interpretation 5**

$$\pi(a) = TRUE$$
$$\pi(b) = FALSE$$
$$\pi(c) = TRUE$$
$$\pi(d) = FALSE$$

- **Interpretation 6**

$$\pi(a) = FALSE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = FALSE$$

- **Interpretation 7**

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = FALSE$$

- **Interpretation 8**

$$\pi(a) = FALSE$$
$$\pi(b) = FALSE$$
$$\pi(c) = FALSE$$
$$\pi(d) = TRUE$$

- **Interpretation 9**

$$\pi(a) = TRUE$$
$$\pi(b) = FALSE$$
$$\pi(c) = FALSE$$
$$\pi(d) = TRUE$$

- **Interpretation 10**

$$\pi(a) = FALSE$$
$$\pi(b) = TRUE$$
$$\pi(c) = FALSE$$
$$\pi(d) = TRUE$$

- **Interpretation 11**

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = FALSE$$
$$\pi(d) = TRUE$$

- **Interpretation 12**

$$\pi(a) = FALSE$$
$$\pi(b) = FALSE$$
$$\pi(c) = TRUE$$
$$\pi(d) = TRUE$$

- **Interpretation 13**

$$\pi(a) = TRUE$$
$$\pi(b) = FALSE$$
$$\pi(c) = TRUE$$

$$\pi(d) = TRUE$$

- **Interpretation 14**

$$\pi(a) = FALSE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = TRUE$$

- **Interpretation 15**

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = TRUE$$

$$16\ Interpretations\ Total$$

For this question, I did not include the evaluations of the knowledge base clauses in my solution. This is because the book defines an interpretation as $I =< D, \phi, \pi >$ and does not depend on a knowledge base, but rather depends on a world or language. Regardless of the construction of the knowledge base, as long as a single language is used, there will still be the same number and format of interpretations. This confusion caused me to exclude the evaluation of the knowledge base with respect to the interpretations. I want to point out that my exclusion of those evaluations is not because I lack the understanding of how to evaluate them, but rather is a result of my misunderstanding of the desired format.

Part 2:

- **Interpretation 15**

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = TRUE$$

$$1\ Model$$

Part 3:

Yes, a is a logical consequence of KB because a is true in all models of KB.

Question 2

Part 1:

The π of each predicate maps to either $TRUE$ or $FALSE$. Therefore, since there are 7 predicts, a, b, c, d, e, f, g, the number of interpretations is as folows:

$$2^7 = 128\ interpretations$$

Part 2:

If the π of every predicate maps to $TRUE$ then all elements of the knowledge base evaluate $TRUE$, thus the interpretation is a model.

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = TRUE$$
$$\pi(e) = TRUE$$
$$\pi(f) = TRUE$$
$$\pi(g) = TRUE$$

Part 3:

If we make the atoms that are 'facts' in KB $FALSE$ in the interpretation then by definition the interpretation is not a model, since those facts evaluate to FALSE in KB.

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = FALSE$$
$$\pi(d) = TRUE$$
$$\pi(e) = TRUE$$
$$\pi(f) = FALSE$$
$$\pi(g) = TRUE$$

Part 4:

Using the bottom up method:

$$\{\}$$
$$\{c\}$$
$$\{c, f\}$$
$$\{c, f, d\}$$
$$\{c, f, d, b\}$$

Therefore the logical concequences of KB are:

$$\pi(c) = TRUE$$
$$\pi(f) = TRUE$$
$$\pi(d) = TRUE$$
$$\pi(b) = TRUE$$

Part 5:

As a result of part four, a, e and g are not logical consequences of KB.

Question 3

Part 1:

Our language has the following attributes:

- Constant symbols a, b and c.
- Oneary predicate symbols p and q.
- Domain of individuals w, x, y and z.

To calculate the number of interpretations can use the following equations:

$$D^c * (2^p)^D * (2^q)^D$$

Where D is the number of individuals in the domain, c is the number of constants in the languages, and p and q are the number of inputs to the p and q predicates respectivly. This evealuates to:

$$4^3 * (2^1)^4 * (2^1)^4 \\ = 16,384$$

Part 2:

Reguardless of what $\phi(a)$ maps to there should be an equal number of $\pi(\phi(a))$ that evaluate to $TRUE$ as $FALSE$. Therefore, there should be the origional number of interpretations divided by two models of this KB.

$$\frac{16,384}{2} = 8,192$$

Question 4

> This question contains several formatting discrepencies that I appologise for. Again I think that the differences between my solution and the provided solution are not a result of a lack of understanding of the material.

Part 1:

Bottom up approach:

$$\{\} \\ \{c\} \\ \{c, e\} \\ \{c, e, b\} \\ \{c, e, b, a\} \\ \{c, e, b, a, j\}$$

The logical concequences are:

$$\pi(c) = TRUE \\ \pi(e) = TRUE$$

$$\pi(b) = TRUE$$
$$\pi(a) = TRUE$$
$$\pi(j) = TRUE$$

Here I failed to include the operation that was being utilized along side the updated set. I did not realize that this was a requirement of the solution. Update as follows:

Rule Applied	Consequent set
.	$\{\}$
c	$\{c\}$
e	$\{c, e\}$
$b \leftarrow e$	$\{c, e, b\}$
$a \leftarrow b \wedge c$	$\{c, e, b, a\}$
$j \leftarrow a \wedge b$	$\{c, e, b, a, j\}$

Part 2:

Model where f is false:

$$\pi(c) = TRUE$$
$$\pi(e) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(a) = TRUE$$
$$\pi(j) = TRUE$$
$$\pi(d) = FALSE$$
$$\pi(f) = FALSE$$
$$\pi(g) = FALSE$$
$$\pi(h) = FALSE$$
$$\pi(k) = FALSE$$

In this section I formatted to solution to reflect that, since f is not a logical concequence of KB we can have a model where all logical concequences of KB are set to TRUE and all non logical concequences (including f) are set to FALSE and have a modle. While this was obvious in my head at the time, it was not when I read back through the problem. The following is just a reorganization of the inital solution to be in alphabetical order by atom:

$$\pi(a) = TRUE$$
$$\pi(b) = TRUE$$
$$\pi(c) = TRUE$$
$$\pi(d) = FALSE$$
$$\pi(e) = TRUE$$
$$\pi(f) = FALSE$$
$$\pi(g) = FALSE$$
$$\pi(h) = FALSE$$
$$\pi(j) = TRUE$$

$$\pi(k) = FALSE$$

Part 3:

Top down method:

$$yes \leftarrow a$$
$$yes \leftarrow b \wedge c$$
$$yes \leftarrow b$$
$$yes \leftarrow e$$
$$yes \leftarrow .$$

> The same thing as Part 1 of this problem:

Answer Clause	Rule Applied
$yes \leftarrow a$	a ← b ^ c
$yes \leftarrow b \wedge c$	c
$yes \leftarrow b$	b ← e
$yes \leftarrow e$	e

$$yes \leftarrow .$$

Question 5

In [5]:
```
# function to parse input knowledge bases
def parse(atom):
    seen = set()
    while atom != []:

        # print the current state of atom before altering
        print("Currently processing %s" % atom)

        # seperate 'first' element in atom from the 'rest' of the atom
        first = atom[0]
        rest = atom[1:]

        # if 'first' is a list set atom =to contents of 'first' + 'rest'
        if(type(first)==list):
            atom = first[:]+rest[:]

        # if 'first' is not a list, check case of first letter
        else:
            # if uppercase add to 'seen' and set 'atom' =to 'rest'
            if(first[0].isupper()):
                seen.add(first)
                atom = rest
            # if lowercase only set 'atom' =to 'rest
            else:
                atom = rest

    # now print out all of the variables
    for var in seen:
        print("Variable %s" % var)
```

Now we run the code above on an input atom:

In [6]:
```
atom = ["p", "a", ["b", "X"], ["d", "e", "X", ["b", "c"], "Y"]]
parse(atom)
```

```
Currently processing ['p', 'a', ['b', 'X'], ['d', 'e', 'X', ['b', 'c'],
'Y']]
Currently processing ['a', ['b', 'X'], ['d', 'e', 'X', ['b', 'c'], 'Y']]
Currently processing [['b', 'X'], ['d', 'e', 'X', ['b', 'c'], 'Y']]
Currently processing ['b', 'X', ['d', 'e', 'X', ['b', 'c'], 'Y']]
Currently processing ['X', ['d', 'e', 'X', ['b', 'c'], 'Y']]
Currently processing [['d', 'e', 'X', ['b', 'c'], 'Y']]
Currently processing ['d', 'e', 'X', ['b', 'c'], 'Y']
Currently processing ['e', 'X', ['b', 'c'], 'Y']
Currently processing ['X', ['b', 'c'], 'Y']
Currently processing [['b', 'c'], 'Y']
Currently processing ['b', 'c', 'Y']
Currently processing ['c', 'Y']
Currently processing ['Y']
Variable Y
Variable X
```

Question 6

I have the programs to take the input query format as "?a" instead of ["a"]. This seems trivial and can be changed in seconds. Also my function takes KB as a parameter in case of wanting to work with different KBs.a

Part 1: Build the parse() function

In [1]:
```
# random library for predicate resolution
import random
import time

# seed with time for non dupliate runs
random.seed(time.time())

# prove(query, kb) is a function that performs a top-down proof of a query
# with randomly selected choices of next steps.
#
#  Inputs:
#    query: a string of the form "?a" for example
#    kb: the knowledgebase of the form[[h1,b11,b12],[h2,b21]...ect]
#
#  Return: (boolean)
#    True - the operation was successful in validating the query
#    False - the operation faild to validate the query
#
def prove(query, kb):


    # check query input format and parse
    if query[0] != "?":
        print("Input not a query, use ?<predicate>")
        exit()
    else:
        current_clause = ["yes", query[1:]]

    # display the inital clause that results from the querey
    print("current clause: ", current_clause)

    # MAIN LOOP
    # continues replacing 'current_clause' body parts that have heads in the
    # base with the corresponding body in the knowledge base until either th
    # body left in 'current_clause' (success) or there is no option to repla
    # parts in 'current clause' (failure).
    while current_clause[1:]:

        # stores clauses whos head matches left most body element of current
        matches = []

        # search kb for clauses and append to 'matches'
        for clause in kb:
            if clause[0] == current_clause[1]:
                matches.append(clause)

        # if empty fail
        if not matches:
            print("failed at", current_clause)

            # Trigger failure return
            return False

        # if option(s) exists, remove matched body part.
        # select option randomly and append to 'current_clause'.
        else:
            current_clause.pop(1)
```

```
            match =random.choice(matches)[1:]
            current_clause += match[:]
            print("current clause: ", current_clause)

    # completion of the while loop without failure, succesfull
    return True
```

Part 2: Extend the parse() function to 100 iterations with the multiparse() function

In [2]:
```
import sys
save_stdout = sys.stdout

# multprove(query, kb) is a function that performs 100 top-down proof attemp
# unless it runs into a succesful one before the 100 mark.
#
#  Inputs:
#    query: a string of the form "?a" for example
#    kb: the knowledgebase of the form[[h1,b11,b12],[h2,b21]...ect]
#
#  Return: (boolean)
#    True - the operation was successful in validating the query
#    False - the operation faild to validate the query
#
def multiprove(query, kb):

    # allow for 100 attempts at proving the query
    for x in range(100):

        # DISREGUARD: Format output for HW submission
        if x < 5: print("\n\nAttempt ", x+1, "\n")
        if x == 5: sys.stdout = open('trash', 'w')
        if x == 97:
            sys.stdout = save_stdout
            print("\n\n\t...\n\t...\n\t...\n\n")
            print("\n\nAttempt ", x+1, "\n")
        if x > 97: print("\n\nAttempt ", x+1, "\n")

        # perform top down proof and see if it is succesful.
        success = prove(query, kb)

        # if there is a success in any of the iterations the
        # query is proven and the function can retern true.
        if success: return True

    # if all iterations have resulted in failures then the
    # query has not been resolved and the function returns false.
    return False
```

Part 3: Prove '?a'

In [5]:
```
kb = [["a", "b", "c"],
      ["b", "d"],
      ["b", "e"],
      ["c"],
      ["d", "h"],
      ["e"],
      ["f", "g", "b"],
      ["g", "c", "k"],
      ["j", "a", "b"]]


multiprove("?a",kb)
```

```
Attempt  1

current clause:  ['yes', 'a']
current clause:  ['yes', 'b', 'c']
current clause:  ['yes', 'c', 'd']
current clause:  ['yes', 'd']
current clause:  ['yes', 'h']
failed at ['yes', 'h']


Attempt  2

current clause:  ['yes', 'a']
current clause:  ['yes', 'b', 'c']
current clause:  ['yes', 'c', 'e']
current clause:  ['yes', 'e']
current clause:  ['yes']
```

Out[5]:
```
True
```

Part 4: Prove '?f'

In [4]:
```
multiprove("?f",kb)
```

```
Attempt  1

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


Attempt  2

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


Attempt  3

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


Attempt  4

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


Attempt  5

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


        ...
        ...
        ...
```

```
Attempt  98

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']


Attempt  99

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'd']
current clause:  ['yes', 'k', 'd']
failed at ['yes', 'k', 'd']


Attempt  100

current clause:  ['yes', 'f']
current clause:  ['yes', 'g', 'b']
current clause:  ['yes', 'b', 'c', 'k']
current clause:  ['yes', 'c', 'k', 'd']
current clause:  ['yes', 'k', 'd']
failed at ['yes', 'k', 'd']
```

Out[4]: False