CS 560: Homework 1 ¶

Eric Stevens

October 7, 2018

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*

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

Part 1:

Bottom up approach:

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

The logical concequences are:

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

Part 2:

Model where f is false:

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

Part 3:

Top down method:

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

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

Part 1: Build the parse() function

In [45]:
```
# 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 [62]:
```
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 [58]:
```
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[58]:
```
True
```

Part 4: Prove '?f'

In [63]:
```
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', 'd']
current clause:  ['yes', 'k', 'd']
failed at ['yes', 'k', 'd']


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', 'd']
current clause:  ['yes', 'k', 'd']
failed at ['yes', 'k', 'd']


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', 'e']
current clause:  ['yes', 'k', 'e']
failed at ['yes', 'k', 'e']
```

Out[63]:
```
False
```

In [ ]: