Systems Theory - Homework 4

Ryan Spangler

January 31, 2011

Chapter 5

I spent all my time writing this fuzzy logic program so didn't get to the questions for chapter 5. However, I am hoping my fuzzy program is spectacular enough to carry me through.

Fuzzy Excellence

For this problem I wrote some python code to implement fuzzy logic, which turned this question into a straightforward application of this logic.

```
variables = {
    'excellent': {8:0.2,9:0.6,10:1.0},
    'good': {6:0.1,7:0.5,8:0.9,9:1.0,10:1.0},
    'fair': {2:0.3,3:0.6,4:0.9,5:1.0,6:0.9,7:0.5,8:0.1},
    'bad': {1:1.0,2:0.7,3:0.4,4:0.1}
}

class FuzzyVariable:
    def __init__(self, key, membership, context=None):
        self.key = key
        self.membership = membership
        if not context:
            self.context = self.membership.keys()
        else:
            self.context = context

    def keyContext(self, keys):
        self.context = keys
    def contains(self, key):
        return self.membership.has_key(key)

    def member(self, key):
        if self.contains(key):
            return self.membership[key]
        else:
            return 0.0

    def fuzzNot(self):
        notted = {}
        for key in self.context:
            value = 0.0
            if self.membership.has_key(key):
                value = self.membership[key]
            if value < 1.0:
                notted[key] = 1.0 - value

        result = FuzzyVariable('not '+self.key, notted, self.context)
        return result

    def fuzzVery(self):
        veryd = {}
        for key in self.membership.keys():
            veryd[key] = pow(self.membership[key], 2)

        result = FuzzyVariable('very '+self.key, veryd, self.context)
        return result

    def fuzzBinary(self, other, op, combine):
        anded = {}
        for key in self.context:
            value = combine(self.member(key), other.member(key))
            if value > 0.0:
                anded[key] = value

        result = FuzzyVariable(self.key+' '+op+' '+other.key, anded, self.contex
        return result

    def fuzzAnd(self, other):
        return self.fuzzBinary(other, 'and', lambda x,y: min([x,y]))

    def fuzzOr(self, other):
        return self.fuzzBinary(other, 'or', lambda x,y: max([x,y]))
```

```python
class Fuzzy:
    def __init__(self, variables):
        self.variables = {}
        self.keys = set()

        for v in variables.keys():
            self.variables[v] = FuzzyVariable(v, variables[v])
            self.keys = self.keys.union(variables[v].keys())

        for v in self.variables.keys():
            self.variables[v].keyContext(self.keys)

    def variable(self, key):
        return self.variables[key]

theory = Fuzzy(variables)

excellent = theory.variable('excellent')
notExcellent = excellent.fuzzNot()
notBad = theory.variable('bad').fuzzNot()
notVeryGood = theory.variable('good').fuzzVery().fuzzNot()

notBadButNotVeryGood = notBad.fuzzAnd(notVeryGood)
goodButNotExcellent = theory.variable('good').fuzzAnd(notExcellent)
fairToExcellent = theory.variable('fair').fuzzOr(excellent)
```

The last three variables contain the membership values for the composite statements. These are:

not bad but not very good:

{2: 0.3, 3: 0.6, 4: 0.9, 5: 1.0, 6: 0.99, 7: 0.75, 8: 0.19}

good but not excellent:

{6: 0.1, 7: 0.5, 8: 0.8, 9: 0.4}

fair to excellent:

{2: 0.3, 3: 0.6, 4: 0.9, 5: 1.0, 6: 0.9, 7: 0.5, 8: 0.2, 9: 0.6, 10: 1.0}

All of these seem to fit fairly well with their intuitive notions. One of the benefits of this approach is that "fuzzy and" and "fuzzy or" are both almost identical except for or uses max() where and uses min(). This similarity is reflected in the fuzzyBinary() function above which also takes a supplied "combine" function which produces the given result based on the values of the respective membership functions of the two variables being compared. This allows for a whole range of logical functions based on two fuzzy sets beyond just and and or.

For instance, rather than max or min, what if the supplied combination function is multiplication? This is a sort of fuzzy convolution, and could be useful in fuzzy applications.

The next step would be to build a natural language parser for the system which uses whatever linguistic variables are supplied at the beginning to convert statements in natural language into executable statements in the fuzzy logic system.