Friday 18-05-18

Part(1):Polarity Table:

1. Different type of words introduces different polarity effects on trees, I divided them into quantifiers and others.
2. We've agreed on a number of changes to the initial tables and this includes using numbers and multiplication for marking the trees nodes instead of the atoms '+','-' and '.'. ✓

   Quantifiers:

```
polTable(*(universal),-1).
polTable(*(no),-1).
polTable(*(proRef),0).
polTable(*(name),0).
polTable(*(the),0).
polTable(_DEFAULT,+1).
```

   Others:

```
polTable2('not',1,-1,_).
polTable2('without',1,-1,_).
polTable2('lack',2,-1,+1).
polTable2('doubt',2,-1,+1).
polTable2(_DEFAULT,2,+1,+1).
```

3. The other change is for polTable2. Instead of saying this word affects N arguments and then placing the sign for the 1st then the 2nd if any, better to have a single list that contains the affected arguments along with the signs that they are designated to have. For example: polTable3('lack',[dobj:-1,subject:+1]).However, it was a bit tricky for me to implement because some words have different pattern depending on the example. 'doubt' affects (xcomp) in 'John doubts she loves every man.' and 'dobj' in 'she doubts it.' ✘
4. in sentences with 'not', 'not' is given a mark and then its effect is propagated properly. However, its mark is removed on later stages (at applyQuants) because it was causing a problem with constructing our final forms. Do 'not' need to maintain its mark? is there hypo/hyper words for 'not'?

Part(2):Examples you wanted to see marked with poalrity

1.doItAll('I have not got ***any*** friends.',X).

```
pm([.,
    arg(claim,
        A,
        [not:1,
         arg(negComp,
             *([time(tense(present),
                     aspect(B),
                     aux(+),
                     def(C),
                     finite(tensed))]),
             [got:-1,
              arg(dobj, *(universal=0.2), [friend>plural:-1]),
              arg(subject, *(proRef), I:0)])])])
```

final form

```
proRef(A::{I:0,A},
       claim(({[(friend>plural): -1],B}
               => (({[tense(present)],C}
                     & ({(D,C),E}
                         & ({(member,E),#0(E,C,B)}
                             => [[got:-1,
                                  {dobj,B},
                                  {subject,A}],
                                 #0(E, C, B)])))
                   => absurd))))
```

(issue 1) ‘any’ is kind of universal quantifier, but I don’t know the monotonicity status of its arguments. So, for now it’s the deafualt (+1).

```
2.doItAll('most men like women.',X).
pm([.,
    arg(claim,
        *([time(tense(A), aspect(B), aux(-), def(C), finite(tensed))]),
        [love:1,
         arg(dobj, *(existential=10), woman>plural:1),
         arg(subject, *(universal), [man>plural:-1])])])
```

all (↓↑)	every (↓↑)
some (↑↑)	no (↓↓)
not (↓)	most (-↑)

(issue 2) most universal quantifiers are downwards on their first arguments, however 'most' is non-monotone on its first argument and based on the current interpretation of most, *(universal), the 1st argument is going have a negative polarity.

Part(3):Examples of proofs where the matching algorithm consider polarity marks.

Simple one and another with negation 'not'

```
1.doItAll('John loves Mary.',X). → doItAll('John likes Mary?',X)
  Yes
2.doItAll('John does not like Mary.',X). → doItAll('John does not love Mary?',X)
  Yes
/**
To be able to do this proof I had to make some changes that I'm not sure if they are right. The claim
final form looks like:
name(A::{[John:NP:0],A},
     name(B::{[Mary:NP:0],B},
          claim((({tense(C),D}
                  & ({(simple,D),E}
```

```
                    & ({(member,E),#0(E,D)}
                        => [[like:-1,
                             {dobj,B},
                             {subject,A}],
                            #0(E, D)])))
                => absurd))))
```

after anchoring and storing its parts as fact and rules-originally- we get:

```
| ?- listing(fact).
fact({salient,_}).
fact({[name,'John':'NP'],'#1'}).
fact({[name,'Mary':'NP'],'#2'}).
| ?- listing(=>).
{[name,'John'],A}=>{he,A}.
{tense(_),A}&({(simple,A),B}&({(member,B),'#0'(B,A)}=>[[like: -
1,{dobj,'#2'},{subject,'#1'}],'#0'(B,A)]))=>absurd.
```

and based on that, the answer to the query is (no; failed to prove it). So, I tried to do something different for setProblem1

```
setProblem1(A => absurd) :-
    !,
    assert(A => absurd),
    setProblem1(A).
```

now get as facts and rules:

```
| ?- listing(fact).
fact({salient,_}).
fact({[name,'John':'NP'],'#1'}).
fact({[name,'Mary':'NP'],'#2'}).
fact({tense(_),_}).
fact({(simple,_),_}).
```

```
| ?- listing(=>).
{[name,'John'],A}=>{he,A}.
{tense(_),A}&({(simple,A),B}&({(member,B),'#0'(B,A)}=>[[like: -
1,{dobj,'#2'},{subject,'#1'}],'#0'(B,A)]))=>absurd.
{(member,A),'#0'(A,B)}=>[[like: -1,{dobj,'#2'},{subject,'#1'}],'#0'(A,B)].
```

doesn't look right but this what allows me to do the proof?! we've talked about proving absurd before but still its not very clear to me how to handle them specially when it comes to storing the claims as facts and rules

**/

matching Modifiers

3.doItAll('John loves an obese woman.',X) → doItAll('John loves a fat woman?',X).

Yes

We are in a +ve context i.e all nodes are upward monotone. However, modifiers seem to have downward entailing direction in positive polarity context?

On the other hand, in negative context, based on the cuurent algorithm, the following will work.

4.doItAll('John does not love a fat woman.',X) → doItAll('John does not love an obese woman?',X).

Yes

is that right ?

skipping over Modifiers

5.doItAll('John loves a fat woman.',X) → doItAll('John likes a woman?',X).

Yes

issue with Modifiers preceded by adverbs like 'very'

'very' is not going to show up in the final form because its of type 'identity'

```
ts([.,
    arg(claim,
        *([time(tense(present),
                aspect(simple),
                aux(-),
                def(-),
                finite(tensed))]),
        [love,
         arg(dobj,
             *(indefinite),
             [woman>singular,
              modifier(amod, [pretty, modifier(identity, very)]),
              modifier(identity, a)]),
         arg(subject, *(name), [John:NP])])])
```