Friday 16-03-18

Part(1): Sorting quantifier stack ;Scoping:

```
X=
[existential::{[tense(present)],A},
existential::{(simple,A),B},
name::{[Fido:NP],C},
indefinite::{[dog>singular],D}]

=>
Y=
[name::{[Fido:NP],C},
existential::{[tense(present)],A},
existential::{(simple,A),B},
indefinite::{[dog>singular],D}]
```

qsort/4 have been used with **compOperatoer** being the 4th argument as it is defined to compare quantifiers scores leaving quantifiers with equal scores unchanged.

Part(2) Doing proofs.

Steps

1-startConversation. to clear up the minutes+ other stuffs

2-doItAll('a man loves a woman.',X). get the final normal form and add it to the minutes.

3-setProblem(minutes) removes old facts and rules and start a new sets using what's in the minutes. Or, setProblem1 extract fact and rules from the minutes and add them to the existed ones.

4-doItAll('a man loves a woman?',X), tries to prove the query X.

from the current version, we can prove the following examples:

'a man loves a woman.' → 'a man loves a woman?'

'a man loves a woman.' → 'a man likes a woman?'

'a man loves a dog.' → 'a man loves an animal?'

Issues:

Issue(1): breaking down the claims' NFs into facts and rules. The NFs of the above examples are conjuncts, so the adjuncts will be the facts. However, there are examples of NFs that include the implication symbol (=>) or list of parts separated by comma (,). As for NFs with =>, I've tried the example 'a man loves dogs' and added the => part as a rule and then send the LHS and the RHS separately so that they turned into facts and rules as well, *at one point I thought maybe all what we need is add them all as fact no rule since we have our QFF? what do you think?.* Then I tried proving 'a man loves doegs?', At first I've got an error says:

! Instantiation error in argument 3 of atom_concat/3

! goal: atom_concat(_61,' ',_63)

When I commented the atom_concat part, I've got an answer but I don't think the variable constant binding makes sense.

the claim

```
({[tense(present)],#0}
  & ({(simple,#0),#1}
      & ({[man>singular],#2}
          & ([[love, {dobj,A}, {subject,#2}], #1]
              => {dog>plural,A}))))
| ?- setProblem(minutes).
yes

| ?- listing(fact).
fact({[tense(present)],'#0'}).
fact({(simple,'#0'),'#1'}).
fact({[man>singular],'#2'}).
fact([[love,{dobj,_},{subject,'#2'}],'#1']).
fact({dog>plural,_}).
yes

| ?- listing(=>).
[[love,{dobj,A},{subject,'#2'}],'#1']=>{dog>plural,A}.
yes
```

the query

```
({[tense(present)],A}
  & ({(simple,A),B}
      & ({[man>singular],C}
          & ([[love, {dobj,#0(C,B,A)}, {subject,C}], B]
              => {dog>plural,#0(C,B,A)}))))

after anwer
({[tense(present)],#0}
  & ({(simple,#0),#1}
      & ({[man>singular],#2}
          & ([[love, {dobj,#0(#2,#1,#0)}, {subject,#2}],
              #1]
              => {dog>plural,#0(#2,#1,#0)}))))
```

next:

‘John loves pretty Mary’ → ‘John likes Mary’

-approximate unification to skip over modifiers

-deal with definite references and names

‘animals are mortals’ ,’Fido is a dog’ → ‘Fido is a mortal’