Part(1):Storing claims as facts and rules.

The way claims have been broken down into facts and rules, I believe, need to be re-visited. Because whenever there is a claim with an outermost implication either the consequent or the antecedent is constructed of conjuncts and/or other implication(s).

```
| ?- doItAll(all men love food.',X). <--couldn't parse this example even after adding 'food' to dictionary
| ?- doItAll(all men love women.',X).
({[man>plural],A}
  => ({[tense(B)],#0(A)}
       & ({(C,#0(A)),#1(A)}
           & ({(member,#1(A)),D}
               => ({woman>plural,#2(D,A)}
                    & [[love, {dobj,#2(D,A)}, {subject,A}],
                       D])))))
| ?- listing(fact).
fact({salient,_}).
| ?- listing(=>).
{[name,'John'],A}=>{he,A}.
{[man>plural],A}=>{[tense(_)],'#0'(A)}.
{[man>plural],A}=>{(_,'#0'(A)),'#1'(A)}.
{[man>plural],A}=>({(member,'#1'(A)),B}=>{woman>plural,'#2'(B,A)}&[[love,{dobj,'#2'(B,A)},{subject,A}],B]
).
```

So I changed the way we deal with rules as following:

```
setProblem1(A => (B & C)) :-
    !,
    setProblem1(A => B),
    setProblem1(A => C).
```

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

and now we get this:

```
| ?- listing(=>).
{[name,'John'],A}=>{he,A}.
{[man>plural],A}=>{[tense(_)],'#0'(A)}.
{[man>plural],A}=>{(_,'#0'(A)),'#1'(A)}.
{[man>plural],A}=>{(member,'#1'(A)),_}.
{(member,'#1'(A)),B}=>{woman>plural,'#2'(B,A)}.
{(member,'#1'(A)),B}=>[[love,{dobj,'#2'(B,A)},{subject,A}],B].
```

I am not sure of this right!—thinking of transitivity rules—but this is what I attempted to do when SATCHMO failed to prove 'all men like women?'. Which was because, at first, I couldn't match [[like, {dobj,E}, {subject,#3}],#4(D, B)] to [[love,{dobj,'#2'(B,A)},{subject,A}],B] as it was a part of a conjuncted consequent of a rule that was a consequent of a rule itself ☹.Then after tearing that rule apart it worked. On the other hand, when the antecedent of a rule was the complicated part, I wasn't sure what to do about it.

```
| ?- doItAll('John does not love Mary.',X).
| ?- listing(=>).
fact({salient,_}).
fact({[name,'John'],'#1'}).
fact({[name,'Mary'],'#2'}).
| ?- listing(=>).
{[name,'John'],A}=>{he,A}.
{tense(_),A}&({(simple,A),B}&({(member,B),'#0'(B,A)}=>[[love,{dobj,'#2'},{subject,'#1'}],'#0'(B,A)]))=>ab
surd.
```

Part(2):Polarity

-in the current version of matching algorithm will allow the folloing:

'every man loves a woman.' -> **'every** human likes a woman.' although (every) is a downward monotone on its 1st argument. (so this is wrong, we need mark 'man' with a negative polarity to ensure downward monton matching, **but when to do the marking?**)

So, we discussed this and we agreed to re-visit it with more examples. Thus, according to [MacCartney and Manning, 2006], most linguistic expressions may be regarded as upward-monotone semantic functions. However, a number of important linguistic constructions are downward-monotone, including

- **the antecedent of a conditional.** Example: If stocks rise, we win -> If stocks soar, we win
- **negation:** not
- **restrictive quantifiers:** no, few, at most n. Example: few athletes -> few sprinters.
- **restrictive verbs:** lack, fail, prohibit. Example: lack weapons -> lack guns
- **certain adverbs:** without, except. Example: without clothes v without pants, a

A few expressions must be considered non-monotone, including **superlative adjectives** 'prettiest' and quantifiers such as **'most'**.
Certain generalized quantifiers must be treated as binary functions having different monotonicities in different arguments such as **'every'** and **'all'** they are bot (↓↑)

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

I played with (qff) to see what a qff from marked with polarity may look like, and here what I have got:

```
| ?- doItAll('John loves Mary.',X).
name(A::{[John:NP],A},
     name(B::{[Mary:NP],B},
          claim(({([tense(present)],#0),(+)}
                  & ({((simple,#0),#1),(+)}
                      & ({((member,#1),C),(-)}
                          => [[love, {dobj,B}, {subject,A}], C]))))))
after anchoring names
claim(({([tense(present)],#0),(+)}
        & ({((simple,#0),#1),(+)}
            & ({((member,#1),A),(-)}
                => [[love, {dobj,#3}, {subject,#2}], A]))))

| ?- doItAll('John does not love Mary.',X).
name(A::{[John:NP],A},
     name(B::{[Mary:NP],B},
          claim((({(tense(C),D),(-)}
                   & ({((simple,D),E),(-)}
                       & ({((member,E),#4(E,D)),(+)}
                           => [[love, {dobj,B}, {subject,A}],
                               #4(E, D)])))
                  => absurd))))
after anchoring names
claim((({(tense(A),B),(-)}
         & ({((simple,B),C),(-)}
             & ({((member,C),#4(C,B)),(+)}
                 => [[love, {dobj,#3}, {subject,#2}],
                     #4(C, B)])))
```

```
| ?- doItAll('every man loves a woman.',X).
claim((({([man>singular],A),(-)}
  => ({([tense(present)],#19(A)),(+)}
       & ({((simple,#19(A)),#20(A)),(+)}
           & ({([woman>singular],#21(A)),(+)}
               & ({((member,#20(A)),B),(-)}
                   => [[love, {dobj,#21(A)}, {subject,A}], B])))))))

| ?- doItAll('not every man loves a woman.',X).
claim((({([man>singular],#22),(+)}
         => ({([tense(present)],A),(-)}
              & ({((simple,A),B),(-)}
                  & ({([woman>singular],C),(-)}
                      & ({((member,B),#23(C,B,A)),(+)}
                          => [[love, {dobj,C}, {subject,#22}],
                              #23(C, B, A)])))))
        => absurd))

| ?- doItAll('no man loves a woman.',X).
claim(({([man>singular],A),(-)}
        => (({([tense(present)],B),(-)}
              & ({((simple,B),C),(-)}
                  & ({([woman>singular],D),(-)}
                      & ({((member,C),#5(D,C,B,A)),(+)}
                          => [[love, {dobj,D}, {subject,A}],
                              #5(D, C, B, A)]))))
             => absurd)))
```

The above examples just to show what a marked tree may look like. This marking process lacks the use of polarity table where words like 'without' are places and lacks the application of polarity compositional operator.

Part(3):Technical

last week I had a form for ‘John is a human?’, but not anymore; no parse tree although we have ‘human’ as nroot and aroot in the dictionary. ‘John is human.’ works though!