I can't see much difference between

(1) a. John was eating some peaches

b. John was eating peaches

In both cases, eating was going on, there were peaches present, and they were what was being eaten. So it's not an exciting example, but we do need to preserve it.

I think that (2) is more interesting.

(2) a. John eats some peaches

b. John eats peaches

It seems to me that there is a clear difference between (2)(a) and (2)(b). In (2)(a) there is an explicit set of peaches which he eats (and by implicature there are some peaches that he doesn't eat). In (2)(b) there is **no explicit set of peaches**.

But, with no further context I'd say that (2)(a) and (2)(b) denote **sets** of eating events. They say that there are (plural) sets of John-eating-peach events. I'm perfectly comfortable with the idea that sentences report sets of events, with the cardinality of the set denoted by the aspect, in the same way that I'm perfectly comfotable with the notion that all NPs denote sets of things, some of which happen to be singleton sets.

But that gives us a handle (2). They both say that there are sets of eating events: but (2)(a) says that there is a set of peaches which are the objects of these events, (2)(b) says that there is a set of peaches for each event. So it's a scope issue. They're both existentials, but '*some peaches*' outscopes the aspect and '*peaches*' is outscoped by it.

To get a handle on this, we make the aspect explicitly introduce a set of events. That's easy enough – we just add a universal quantifier inside the scope of the existential which we used to get from the aspect. So for

(3) John sleeps.

we used to get (before `qff`, because I think it's easier to see what's going on while we have real quantifiers) (note that reference now outscope `claim`: that's the right thing to be doing – see the end of this document for a brief commentary on how this works).

```
name(A::[John:NP],A,
     claim(exists(B::[tense(present)],B,
                  exists(C::(simple,B),C,
                         [[sleep, subject,A], C]))))
```

and I now get

```
name(A::[John:NP],A,
     claim(exists(B::[tense(present)],B,
                  exists(C::(simple,B),C,
                         forall(D::(member,C),D,
                                [[sleep, subject,A], D])))))
```

({`(member,C),D`} here means `D` is a member of `C`: slightly odd way to write it, but I can do that if I want)

OK, now using this treatment of aspect I can get different interpretations of (2)(a) and (2)(b) that bring out the difference.

(S2) a. John eats some peaches.

```
name(A::[John:NP],A,
     claim(exists(B::[tense(present)],B,
                  exists(C::(simple,B),C,
                         exists(D::[peach>plural],D,
                                forall(E::(member,C),E,
                                       [[eat, dobj,D, subject,A],
                                        E]))))))
```

b. John eats peaches.

```
name(A::[John:NP],A,
     claim(exists(B::[tense(present)],B,
                  exists(C::(simple,B),C,
                         forall(D::(member,C),D,
                                exists(E::peach>plural,E,
                                       [[eat, dobj,E, subject,A],
                                        D]))))))
```

That does nearly everything I want. In (2)(a) there is a set of peaches and a set of events in which John eats that set. Note that since you can only eat a given peach once, that more or less forces the set of events to be a singleton, which might be nice. In (2)(b) there's a different set of peaches for each event in the set, which is definitely what I want.

This does sensible things with

(4) a. John does not eat peaches.

```
name(A::[John:NP],A,
     claim(not(exists(B::[tense(present)],B,
                      exists(C::(simple,B),C,
                             forall(D::(member,C),D,
                                    exists(E::peach>plural,E,
                                           [[do,
                                             dobj,E,
                                             subject,A],
                                            D])))))))
```

b. John is not eating peaches.

```
the(A::tense(present),A,
    name(B::[John:NP],B,
         claim(not(exists(C::(prog,A),C,
                          forall(D::(member,C),D,
                                 exists(E::peach>plural,E,
                                        [[eat,
                                          dobj,E,
                                          subject,B],
                                         D])))))))
```

We still have to do something useful with the aspect markers, but that's stuff we can look at as we go along.

So that was good, but we haven't yet looked at the syllogistc versions.

(5) All cats are idiots.

```
claim(forall(A::[cat>plural],A,
             exists(B::[tense(present)],B,
                    exists(C::(simple,B),C,
                           forall(D::(member,C),D,
                                  exists(E::idiot>plural,E,
                                         [[be,
                                           predication(xbar(v(-),n(+))),E,
                                           subject,A],
                                          D]))))))
```

Well it's very long and unreadable, but it's probaly not wrong (!). If `A` is a set of cats then there's a set `E` of idiots and right now there's a relation between them. A painfully complex looking relation, but a relation nonetheless. A bit of catching things during the preprocessing or postprocessing stages will make this look better.

To get to here I've done various things.

(i) I've made '*not*' do sensible things. Syntactically it takes a sentence as argument and returns a sentence: that's not unreasonable, but it's a bit complicated because I want to make it take the whole tensed sentence, so that it dominates the auxiliaries in things like '*John is not eating peaches*', and that makes things a bit complicated. A bit complicated, but do-able, so we do it.

```
[.,
 arg(claim,
     A,
     [not,
      arg(negComp,
          *(time(tense(present),
                 aspect(simple),
                 aux(+),
                 def(+),
                 finite(tensed))),
          [[be],
           arg(auxComp,
               *(time(tense(present),
                      aspect(prog),
                      aux(-),
                      def(B),
                      finite(participle))),
               [[eat>ing,
                 arg(dobj, *(existential=10), peach>plural)],
                arg(subject, *(name), [John:NP])])])])]
```

(ii) The tree above has a bit that looks like `[not, arg(negComp, ...)]`. We have to deal with this in two places – in `timeSequence`, where it's a bit of a hack, and in `qlf`, where we just get rid of it because it was just something we stuck in the parse tree.

(iii) I've changed `assignScopeScore` so that scope can be assigned locally, e.g. in the lexicon, and then just extracted when we see it. To date I'm just using it for generics, which I'm specifying as `*(existential=10)` – as existentials with scope value 10 (so they end up with very local scope). We can do the same with other things, e.g. if we decide to think of '*any*' as a very wide scope universal we could do it by putting it into the lexicon with specifier equal to `specifier(*(universal=0.2))`.

(6) John is not eating any peaches.

```
the(A::tense(present),A,
    name(B::[John:NP],B,
         claim(forall(C::[peach>plural],C,
                      not(exists(D::(prog,A),D,
                                 forall(E::(member,D),E,
                                        [[eat,
                                          dobj,C,
                                          subject,B],
                                         E])))))))
```

(iv) I'm allowing arbitrary items to have scope. That's a slightly weird sounding notion, but it makes sense. So `not` is a scoped operator, and so are `claim` and `query`. That lets reference outscope the discourse operators, which is also the right thing to do.
There's quite a bit here to talk about. I'm fairly happy with what's here, but we'll need to work our way through it. Especially what we're going to do with the syllogistic versions. See you on Thursday.

(I've stuck in lots and lots more printing so I can see what I'm doing: we can take that out later)