2019-12-08: There is a mistake in the way that the splitting of the root context is implemented.

Consider an ontology where you have O3(x) -> x = o2 v x= o1, and then you are checking a query of the form A ->F such that T->B1(o1) and T->B2(o2) have appeared in the context for A. Clearly, B2(x) -> B2(x) have appeared in the context for O2(x), together with O3(x) ->O3(x).

But now, suppose the ontology makes B2 and O3 incompatible. You derive O3(x) ^ B2(x) -> bot in the context for O2, but how do you propagate this clause to the context with A? You do not have a maximal O3(x) in A. You cannot propagate it either to O3, because you do not have a maximal B2(x).

More abstractly, the problem appears when, in a nominal context, two body atoms proceeding of different contexts interact with each other. In general, those two contexts may be completely unrelated, so it is not bad that clauses obtained from this interaction cannot be propagated, as they are irrelevant (this is a well discussed property of CB reasoning in general, as seen in my thesis). However, the problem raises when one of the atoms in the body is produced by a certain clause i.e. a clause of the form Top -> …

In that case, it should be possible to propagate the clause, at least, to the context where the other body atom comes from. But how to do this?

One possibility is: propagate certain clauses among nominal contexts whenever necessary (for instance, if a nominal mentioned in them is mentioned in a nominal context). This seems more in line to what I have been doing so far.

Thus, I would propagate Top -> o3=o1 v o3=x, and since o3=o1 is inert, I would end up with this atom o3=o1 in the root context. Then, this should trigger something that continues the derivation. Or better; propagate equalities as O3(o1), to avoid having to add new types of atoms.

NOW the question is, do I just implement this as it is, or should I somehow wait until I do the massive re-factoring for triggers?

Pros of now: have a presumably sound, complete thing ASAP, which is good in case unexpected stuff happens.

Cons of now: may take longer. There may be more mistakes that may be revealed if I do the refactoring, so maybe prioritising the refactoring is better.

The last argument convinces me; I will prioritise the unified strategy and the refactoring.

After having thought about the underlying strategy, I conclude that the previous already was essentially correct, the only problem was on application of Eq rules, since unlike the case of Coll, to apply equalities to B(x) and x = o, contrarily to the case of query contexts, you cannot assume that B(x) is eventually derived from the core A(x), so you have to either make a new Coll rule, nColl, which works like Coll for these atoms B(x) that are not in the core, or you just apply Eq followed by r-Succ. The second case seems to be better in this case because, if we use nColl, the corresponding nQueryPred rule may involve arbitrarily many clauses, and this is bad. Other inferences seem always possible.

Ok, but this has the problem that different body atoms in a relevant clause may proceed from different contexts. So the r-Pred is not necessarily applicable. Especially if one of them is the query context, such clause cannot be propagated to any other context.

So, originally, all r-Preds use maximals from the query context. So to ensure that there are no problems, going back with how to deal with equalities B(x) and x=o, doing the r-Succ thing is important to ensure completeness, but also it creates the problem that I can do r-Pred with body atoms B(o) and those of other context. So, to prevent this from happening, one thing I can do is “import” the full clause I would be doing r-Pred with in the nominal context. So if I need to r-Pred a conclusion involving B(o) into a query context, I do that through this new clause, corresponding to the Gamma -> Delta v B(o) that I had in the other nominal context. Equalities are transformed from x=a to Oa(x). R(y,x) stays the same.

NOTICE that this solution is a general case of the one I considered above.

But this has a problem: r-Pred cannot be actually triggered if Gamma contains a clause C(a), since there is no link labelled a from the query context to nominal context for o.

And I think the answer needs to be: propagate even without the link.

Or the other option is to propagate the links. You don’t just copy the clause; you also copy any incoming links from any query context, so that this can be propagated directly to each relevant query context.

OK, so in conclusion, to fix the problem, it is better to, in addition to r-Succ, copy the whole clause and links to the other nominal context, then proceed as usual.

Perhaps it is better to make just one root context for all nominals. But this has the problem of the R(o,f(o)) atoms. A possible solution is to use a single root context, and then have a rule that works like Succ, but with atoms in o, instead of f(x), and then maps to contexts where consequences can be explored. For instance, Gamma -> Delta v B(o) could trigger something to the context B(x). It is a limited form of lateral reasoning. At the end of the day, this is just about what is the best way of splitting the inferences, and the answer tends to be that the more goal-oriented, the better. So maybe I should just look for ways of making it more goal oriented.

The advantage of having a single root context is that it is more clear conceptually what is happening.

The disadvantage is: 1) having clauses R(o,f(o)); too many types of terms in a context. A potential solution has been delineated above.

2) Bad for parallelisation. Many contexts accessing the same context at the same time. But is that really true? Query contexts will access it, for sure. Non-query contexts will also access it for reasoning with R(x,o).   
  
It “could” be true. You are really interested in how the deductions in the root context change each time you add a new ground atom from a query context, but you are NOT interested in such ground atoms coming from different query contexts interact together. This can be particularly problematic.

So ideally, you want to ``split’’ the root context as much as possible. But is that really true? No, most inferences in the root context happen automatically, constant communication is not really a problem. The REAL PROBLEM is that you do not want to be doing too many calculations within the root context.

So one way of avoiding this is by breaking the root context into a context for each nominal. This is the way Sequoia is currently implemented.

But you have to be mindful of what happens when the different parts need one another to do some inference; this leads to the issue I discussed above. I just need to be clearer on why this breaking up is equivalent. Maybe I should write something up.

Is there some other way of breaking up the root context? Yes: add a ``sub-root’’ context for each query context. You then split reasoning not by constants, but by queries. But then, to whom connect the non-query contexts? To all sub-root context it has an ancestor to. Or alternatively, we do not rely on the nominal/root contexts to do this, bc at the end of the day, it is just a resolution step always. This may also provide a natural integration with the alternative calculus, maybe. It is more general than it, but it is closer.

Another option is to break it up by context. As long as I have a good way of importing the parts I need from the root context (remember, I care about parallel computation, multiple accesses to root context are not necessarily bad), this should work. As always, however, the problem is whether this ``splitting’’ will lead to just repeating inferences over, for instance, if multiple non-query contexts have the same non-certain ground atoms.

Finally, you could attempt to split the context in both ways: by nominal, and by query context.

Idea: just write paper PARALLEL REASONING IN SEQUOIA (and include Datatypes)

Split-by-nominal appears to be one way. Other ways can be trickier. Maybe I should perfect lateral reasoning; if I have Gamma1 -> Delta1 v C(o), then Gamma2 -> Delta2 v D(o) in the same context, then I add a new sub-root for C(o), and another for D(o). But then if in the context for C(o) I happen to derive D(o) **FOR SURE** then I merge the context C(o) into that for D(o).

Then again, I am worrying about stuff that may not actually be problematic. So what I suggest for future action is: fix implementation for nominal split, and write the theory to ensure no unexpected things happen.

Then refactor code (it needs doing regardless).

Then look at performance bottlenecks, and bear in mind that you can always implement the one-root context version, and maybe extend that in the future with parallel reasoning, including for root thingies.

What if the expansion strategy is a SMARTER MANAGER? It could keep track of lateral reasoning stuff.

End 2019-12-08.