Join

Implementation:
For a join query of this format: **join (** xq1 **,** xq2 **,** attrList1 **,** attrList2 **)** our implementation of the join can be divided into three parts:

1. Visit xq1 and get a list of result nodes. For each attribute in attrList1, create a hashmap <String, ArrayList<Node>> using string value of attribute as name and a list of corresponding nodes as value.

2. Visit xq2 and get a list of result nodes. For each node check if each attribute value in attrList2 exists in the hashmap that previously created for the corresponding attribute in attrList1. If so, get the node from the hashmap and add it to a list as a temporary result.

3. If the temporary result list is not empty, build new nodes by combining current node and the nodes in the temporary result list. Append the children of current node to each node in the temporary result list. Return the list of new nodes after finish building all nodes.

Obesrvation:

1. We noticed that attributes list could be empty. That means the two sets we need to join do not have common attributes. In that case, we should do Cartesian product between the two sets. We simply build new nodes for each pairs of node in the two sets by appending children of the right set to the left set.

2. We also noticed some interesting feature of g4 grammar when introducing the new return format for join clause, for example:
   "*return<triplet>{$tuple/b1/*,$tuple/b2/*,$tuple/b3/*}</triplet>"*
   When parsing this query, it result in a wrong tree which parse the whole *"$tuple/b1/*,$tuple/b2/*,$tuple/b3/*"* as one xpath, without parsing it to three xpath separated by commas. This can be fixed by changing the order of xquery types in g4. After we switched the "xq, xq" with "xq / rp ", the query could be correctly parsed.

ReWrite

Implementation

A. Divide sets

We divide variables into different sets so that the variables in a certain set do not have dependence relation with any variables in other sets. Dependence relation is defined as :

for each sentence “$var in xq” in FLWR query, we say $var have dependence relation with the variables appears in xq. If xq contains keyword “doc” , we create a new set and put all related variables in this set.

In order to keep this dependence relation, we also need to maintain the order of variables within each set, which is also the appearance order for each sentence. We will keep this order and use this order when we re-write the “For” and “Return” query.

B. Use conditions in whereClause to connect sets to be joined
In each iteration we choose two sets from the sets list and merge them. Repeat this until only one set left.

For a where condition of the format “var1 eq var2”, we need to consider two situations:

1. Var1 and Var2 in the same set.
2. Var1 and Var2 belongs to different set.

For the first situation, join operation cannot be done, so we need to maintain this where condition under this set. For the second situation, we can do a join between the two sets where var1 and var2 belong. We accomplish it by storing two vars, which belongs to two different set, in two different bags. We merge the two sets and build join query for this merged set.

We have two list of strings, one called createSentence to store join clauses for each set, another called whereSentence to store remaining where clauses for each set. In each iteration, if previous createSentence for this set is empty, which means it is the first time to use this set, create a FWR xquery for constructing this set, otherwise use the previous generated constructing query.

In each join query, two xquery respectively comes from the queries which constructed this set. Two attribute lists comes from two bags which store all variables which belongs to the second situation.

C. Build return sentence
After rewriting all the joins and appending them together with remaining where clause, we build the last return clause. We replace all the variables “varName”in the original return clause with $tuple/varName/*.

Obesrvation:
When all the conditions in the original where clause has been used up, there are situation when there are more than one sets left. If that happen, we make two attributes lists empty, so that can do Cartesian product when they join.