BROWN UNIVERSITY
Department of Computer Science
Master's Project

CS-96-M8

"Optimizing Predicates in Object-Oriented Queries"

by

Yung-Ming Chien

Master Project Report

Optimizing Predicates in Object-Oriented Queries

Author: Yung-Ming Chien
Advisors: Marian H. Nodine
Stanley B. Zdonik

May/1996

Table of Contents

Master Project Report

Optimizing Predicates in Object-Oriented Queries

Author: Yung-Ming Chien
Advisor: Marian H. Nodine
SISW86693
Sept/1994~May/1996

1. Introduction: Project Description.

In my project, I am focusing on optimizing predicates in the EPOQ object-oriented query optimizer. Since often more than one query plan could satisfy the same query, I developed strategies to find more efficient query plans for evaluating predicates. Right now, this project consists of the EPOQ regions: "normalize", "select", "apply", "andnot", and "CNF"(also known as Conjunctive Normal Form). Under the "CNF" region, there are two other regions: "Simplify AND" and "Simplify OR". I developed the "normalize", "CNF", "Simplify AND", and "Simplify OR" regions. Section 3 is an overview of these regions.

2. Overview of EPOQ.

In [1], Mitchell introduced an extensible architecture for a query optimizer in an object-oriented database. It modularizes different query optimization strategies into separate optimization regions, and uses an overall control strategy to determine which regions are applicable to a specific query, and in what order.

The EPOQ query optimizer takes a modular approach to optimization. It consists of a tree of regions, each of which does a specific optimization task, called a "goal". A region may attain its goal either by transforming the query itself or by calling different child regions to transform it.

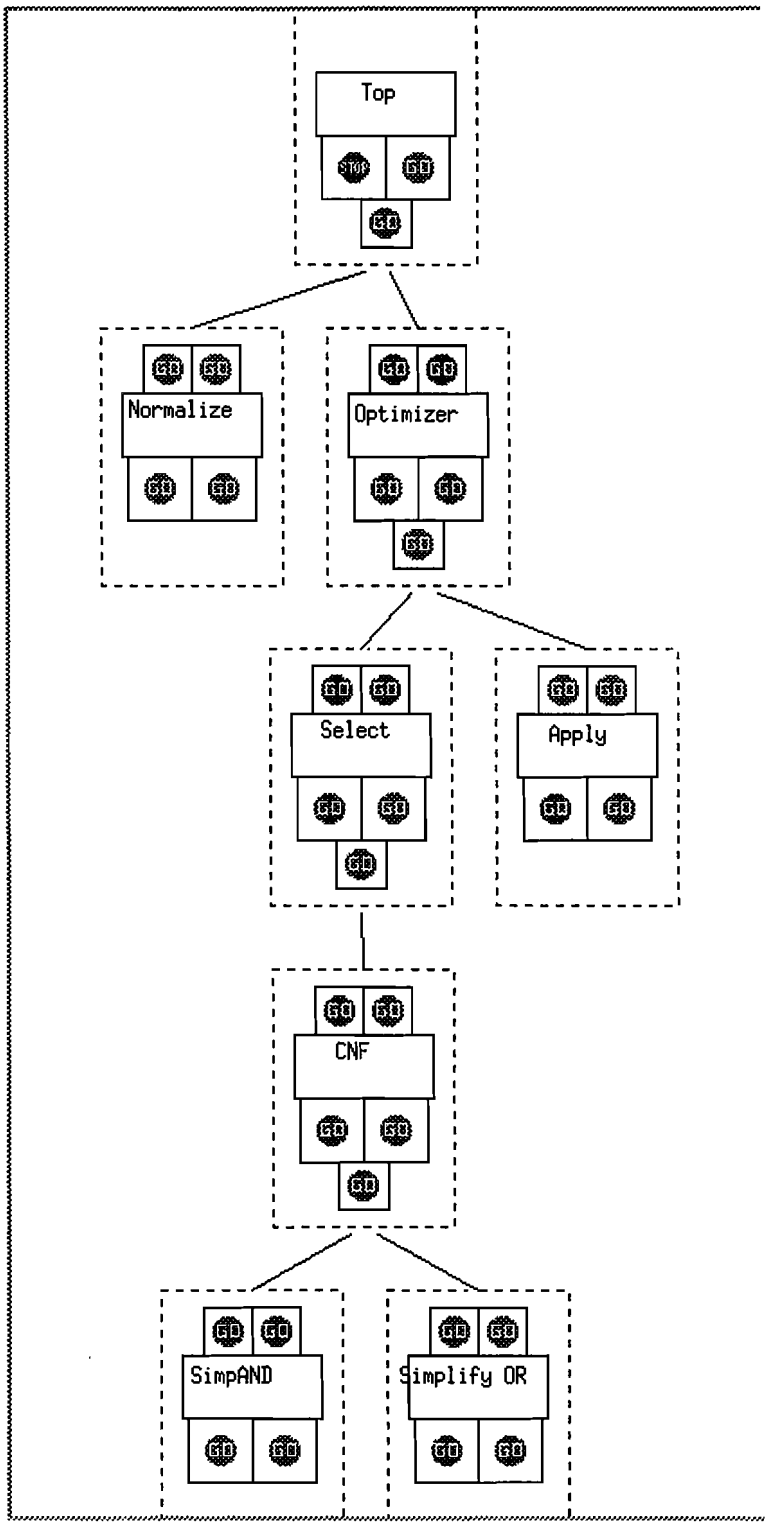
In each region, there are two major methods which are CustomApplicable and CustomAttain. These methods are used to interface between regions. CustomApplicable examines an input EAT as to whether the EAT is of a form the region can process. It returns a "bid" which indicates how much the region thinks it can improve the query. A parent region will call CustomApplicable on all child regions that it has for this goal, then compare the bids to determine which will be best. It then calls CustomAttain on the best region. CustomAttain does the actual transformation. It does the actual transformation of the EAT. It is the core method of the region.

Queries in some object-oriented query language are parsed into an internal representation based on the AQUA query language[1]. Initially, the optimizer will only be able to parse AQUA queries into the internal representation, though we plan to look at other languages as well in the future.

The internal representation of a query is passed from the parser to a top-level control region for the optimizer itself. The top-level region does no optimization, but does pass the query representation to other, appropriate regions which actually do the optimization work. We support a tree of optimization regions, where each region coordinate the execution of several subregions.In this project, I implemented four subregions: "normalize" region, "CNF" region, "Simplify AND" region, and "Simplify OR" region. Each of them considers different specific optimizing cases. They are used to optimize predicates in an object-oriented query.

Figure 2.1 is the snapshot of the current EPOQ regions. The flow of control is as follows: The query representation is sent to the "Top" region. The "Top" region first sends the query into "normalize" region to normalize the predicates. The normalized query is returned to the "Top" region, which then sends it to the "Optimize" region. Thirdly, the "Optimize" region decides which child is the best(based on some query traversal strategy and the results of CustomApplicable) and sends the query into it. If the query is a select, it is sent into the "select" region. After doing some transformations to bring the select predicates together, the query is sent into the "CNF" region to transform the query to CNF. Then the "Simplify OR" is called to do some simplification and reordering. The conjunct is then sent to the "SimpAND" region and do some simplification and reordering. The new query is sent back to "CNF" region, "Select" region, and "Optimizer" region. If the query contain "apply" operator, it is then sent to the "Apply" region. The new query is then sent back to "Top" region, because the optimizer is done.

Fig. 2.1



Within the optimizer, a query is represented as an Extensible Annotated Tree(an EAT). EATs were first developed for an optimizer for EQUAL queries[1].

An EAT is composed of alternating layers of function nodes and data nodes, connected by labeled arcs. Data nodes represent data that is manipulated during the execution of the

query. It can represent either an object in the database(if it is a leaf in the EAT).; or an object built by a query, subquery, or other function (if it is an internal node in the EAT). The root of the EAT is a data node representing the result of the query. Function nodes represent actions that can be taken on data. Its children represent the inputs to the function. Thus, a function node always has at least one child node. A function node also always has one parent data node, representing its output. Function nodes are represented by boxes with the function name contained inside. Data nodes are represented by ovals with the data type inside. Fig. 2.2 is the example of EAT, the AQUA query is 'apply(lambda(p) p.name)(People)'.

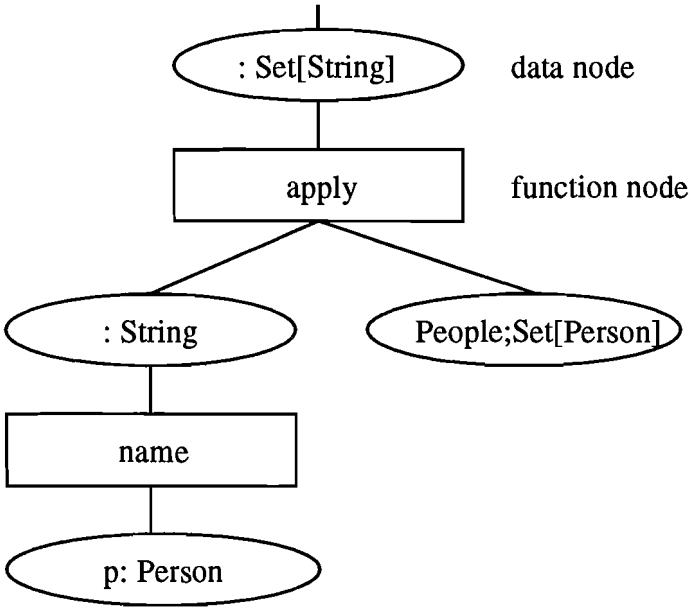


Fig. 2.2 Example of an EAT. The AQUA query is:

apply(lambda(x) p.name) (People)

3. Region Overview.

This section summarizes the regions which are implemented in my project. Section 3.1 describes the "normalize" region. Section 3.2 describes "CNF" region. Section 3.3 describes "Simplify AND" region. Section 3.4 describes the "Simplify OR" region.

3.1"normalize" region

- **Purpose:**

Eliminate NOT nodes where possible, keep variables for comparators on the left-hand side, where possible. Also, convert obviously trivial comparisons to TRUE or FALSE

predicates.

- **Overview:**

Users specify a query in a way that is logical to them. But there may be some redundancy in the query. The "normalize" region tries to eliminate any redundancy in the predicate. Also, it puts simple (comparison) predicates in a normal form to simplify processing later. Basically, this region does three things. First, if the predicate has a single "not" over a comparison, then it deletes the "not" and change the sense of the compression. For example, if the query looks like this:

select(lambda(x) not(x.stars != 4))(Hotels),

then after normalizing, the predicate will become this:

select(lambda(x) (x.stars = 4))(Hotels).

Secondly, if the predicate is of the form (not(not p)), then it deletes both "not"s. For example, the input query is:

select(lambda(x) not (not (x.stars > 4)))(Hotels)

then after normalizing, the predicate is:

select(lambda(x) (x.stars > 4))(Hotels).

Thirdly, if there is a constant on the left-hand side of the predicate, then it changes it to be the equivalent predicate with the constant on the right. For example,

select(lambda(x) 4 <= x.stars)(Hotels)

will be transformed to:

select(lambda(x) x.stars>=4)(Hotels)

This ensures that any predicate with one variable has that variable on the left-hand side. This simplifies future processing, as later regions can assume that all variables and path expressions for subpredicates that do comparisons are on the left, except when the subpredicates compare two variables or path expressions.

3.2 "CNF" region.

- **Purpose:**

Transform the input query to Conjunctive Normal Form (CNF). A comparison involving constants, variables, and path expressions in a boolean formula is an occurrence of a

variable or a path expression. Once the query is in CNF, use the 'Simplify OR' and 'Simplify AND' regions to optimize the conjuncts and disjuncts.

- **Overview:**

CNF, is also known as Conjunctive Normal Form. A comparison involving constants, variables, and path expressions in a boolean formula is an occurrence of a variable or a path expression. A boolean formula is in Conjunctive Normal Form if it is expressed as an AND of clauses, each of which is the OR of one or more comparison involving constants, variables, and path expressions. This region tries to normalize the input query to Conjunctive Normal Form. For example, the expression ((a AND b) OR c) is transformed to CNF, it will be ((c OR a) AND (c OR b)) The boolean operators AND and OR are separated. So, if the input query is:

select (lambda(x) (x.stars!=2) AND (x.stars>=3) OR (x.stars<2)) (Hotels),

then when transformed to CNF it is:

select (lambda(x) (((x.stars<2) OR (x.stars>=3)) AND ((x.stars<2) OR (x.stars!=2))) (Hotels).

After we got the CNF query, then each subquery is sent to the "Simplify OR" region to be optimized individually. Then the whole new query into "Simplify AND" region to simplify and optimize the conjunct. Finally, we return the optimized predicate.

3.3 "Simplify AND" region

- **Purpose:**

Optimize predicates that are ANDed together. Eliminate redundancy, simplify the predicates algebraically, and sort the predicates into rank order[2] for optimal execution speed.

- **Overview:**

This region optimizes predicates that are ANDed together. It eliminates duplicate predicates. It also simplifies the predicate algebraically by simplifying groups of predicates that are defined over the same path expression. Finally, once the predicate is simplified, it orders the conjuncts in rank order[2] to optimize the time required to evaluate the predicate. Rank order is using the "rank" to order the predicates. The "rank" is selectivity minus one then divided by the cost.

$$RANK = \frac{(SELECTIVITY - 1)}{COST}$$

The cost of invoking each selection predicate on a tuple is estimated through system metadata. The selectivity of each selection predicate(i.e. the percentage of tuples expected to satisfy the predicate.) is similarly estimated and selections over a given relation are

ordered in ascending rank order. According to [2], this ordering is easily shown to be optimal for selections. The lower the selectivity of the predicate, the earlier we wish to apply it, since it will filter out many tuples. Similarly, the cheaper the predicate, the earlier we wish to apply it, since it doesn't cost much to evaluate it.

Examples of the "Simplify AND" region are as follows: if x.stars is an integer, and the input query is:

select (lambda(x) x.stars>3 AND x.stars>4 AND x.stars<5) (Hotels),

then x.stars>3 and x.stars>4 can be reduced to x.stars>4. x.stars>4 AND x.stars<5 will return FALSE. So when the example-query sent into "Simplify AND" region and after sorting into rank order, it will return the new query:

select (lambda(x) false) (Hotels)

Give an example of rank ordering:

select (lambda(x) x.stars>=4 AND x.name = "Hilton") (Hotels)

is transformed to

select (lambda(x) x.name="hilton" AND x.stars>=4) (Hotels)

because both subpredicates have the same cost, bur there are fewer hotels with the name "Hilton" than there are 4 and 5 star hotels. Thus, it is more efficient to evaluate "x.name=Hilton" first, because if it is false you don't even need to examine the number of stars.

3.4 "Simplify OR" region

- **Purpose:**

Optimize predicates that are ORed together. Eliminate redundancy, simplify the predicates algebraically, and sort the predicates into inverse rank order. Inverse rank order is the opposite of the rank order specified in [2].

- **Overview:**

This region optimizes predicates that are ORed together. It operates in a similar way to the "Simplify AND" region but uses different methods to simplify the predicate algebraically. Also, once the predicate is simplified, it uses inverse rank to ensure optimal evaluation speed. We use inverse rank for OR predicates because we want to evaluate the subpredicates first that are most likely to be TRUE. Let's back to the example mentioning in "CNF" region section:

select (lambda(x) (((x.stars<2) OR (x.stars>=3)) AND ((x.stars<2) OR (x.stars!=2)))

(Hotels),

then we sent the subqueries ((x.stars<2) OR (x.stars>=3)) and ((x.stars<2) OR (x.stars!=2)) into "Simplify OR" region. ((x.stars<2) OR (x.stars>=3)) will transform to (x.stars!= 2). ((x.stars<2) OR (x.stars!=2)) will transform to TRUE. So the new query returning from "Simplify OR" region will be:

select (lambda (x) (x.stars != 2) AND true) (Hotels)

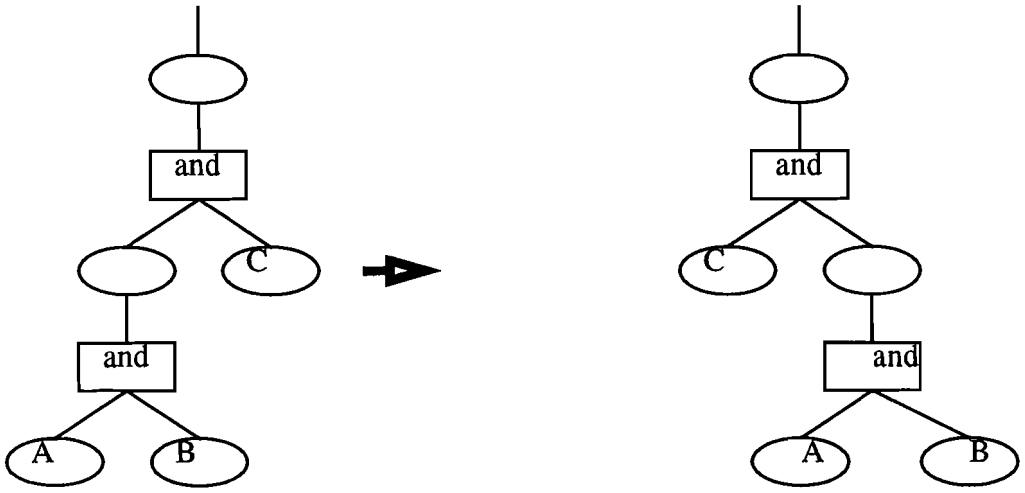
Then we send the new query we got from above procedure: ((x.stars != 2) AND true) sending into "Simplify AND" region, then we got the new, simplified query as following:

select (lambda(x) x.stars != 2) (Hotels)

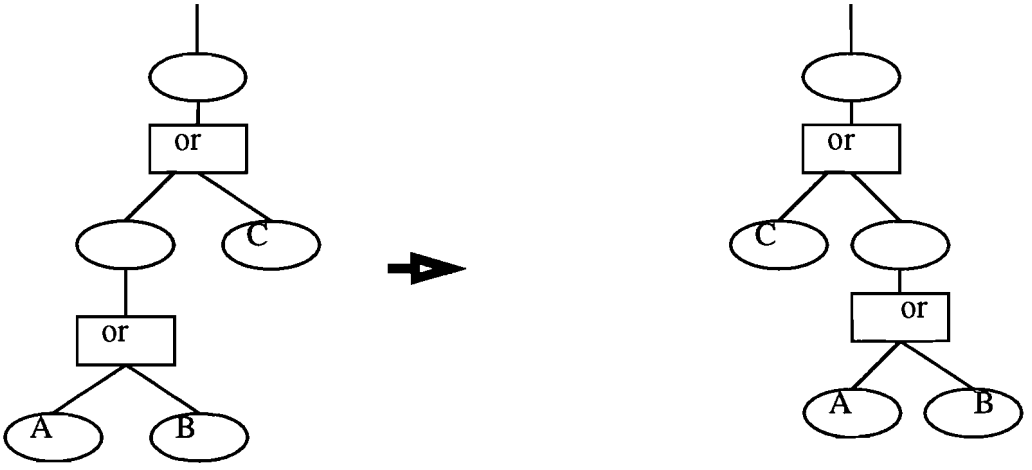
4. Transforming rule to Conjunctive Normal Form

In general, a boolean formula contains bunch of ANDs and ORs operators. A comparison involving constants, variables, and path expressions in a boolean formula is of a variable or a path expression on the variable. A boolean formula is in Conjunctive Normal Form if it is expressed as an AND of clauses, each of which is the OR of one or more comparison involving constants, variables, and path expressions. According [3], there are some rules to transform the general boolean formula to CNF. These rules are sufficient to transform the general boolean formula into CNF. Rule 1 to rule 6 are the transforming rules to transform the EAT in CNF. Rule 7 to Rule 9 are the simplified rule to simplify the EAT in CNF and we don't use the individual method to implement the rule8 and rule 9 but they are handled by conditional statement. The Other cases are: (A and A) --> A and (A or A)-->A. These are considered in Simplify AND(OR) region.

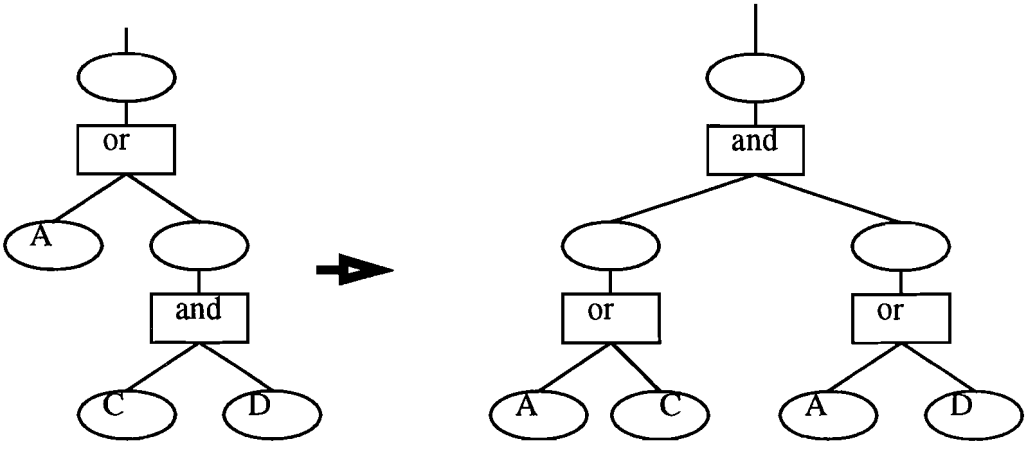
Rule 1: (A and B) and C --> C and (A and B)



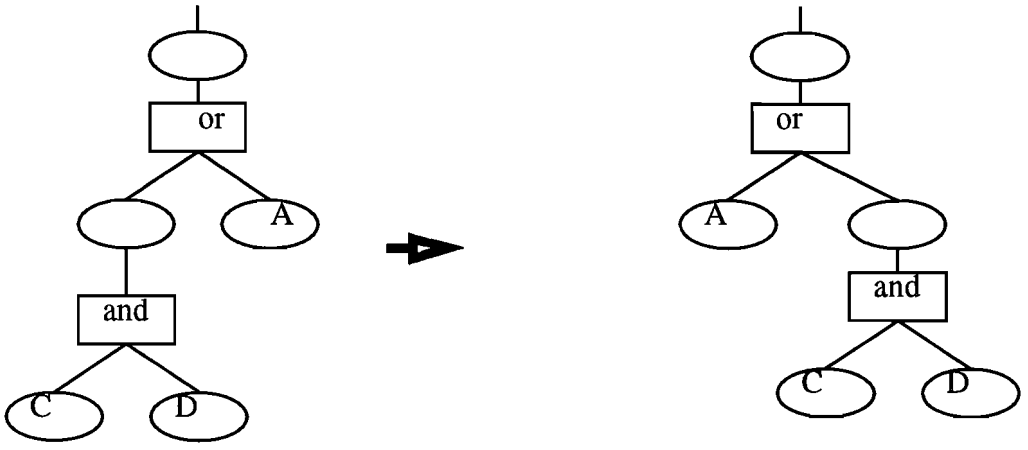
Rule 2: (A or B) or C --> C or (A or B)



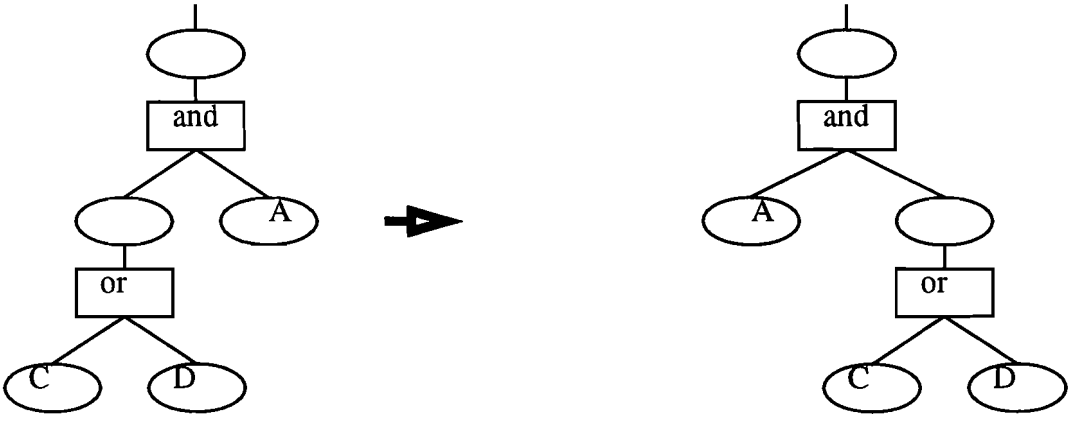
Rule 3. A or (C and D) --> (A or C) and (A or D)



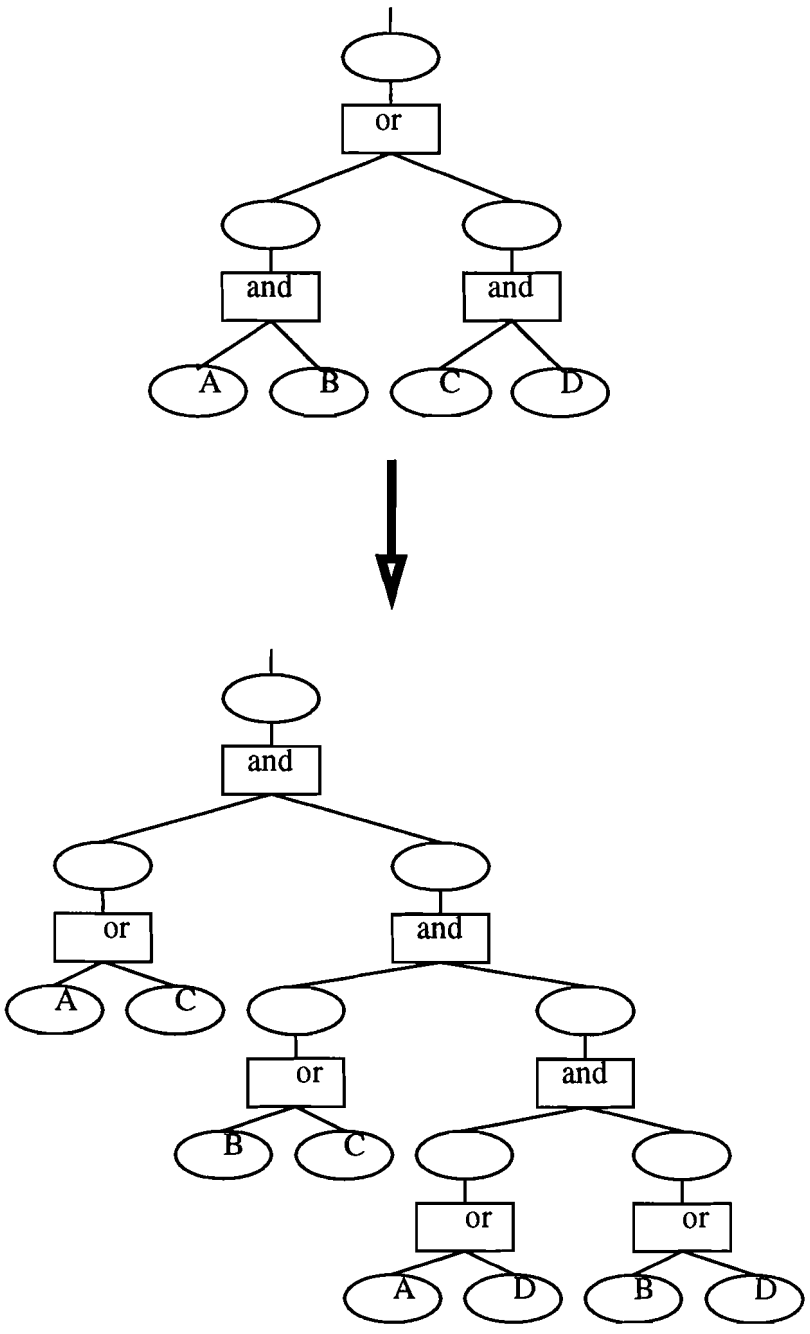
Rule 4: (C and D) or A --> A or (C and D) --> Rule 3



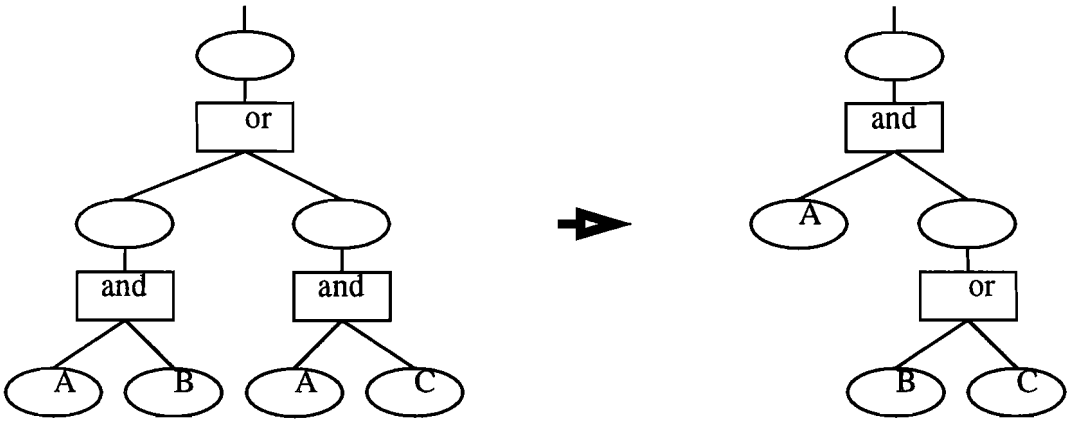
Rule 5: (C or D) and A --> A and (C or D)



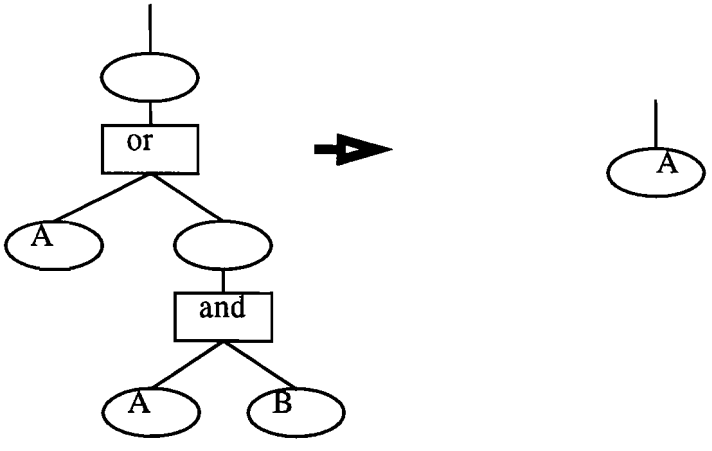
Rule 6 :(A and B) or (C and D) --> (A or C) and ((B or C) and ((A or D) and (B or D)))



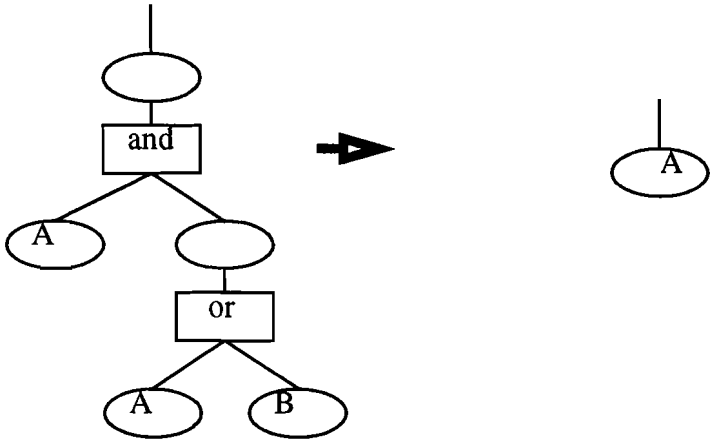
Rule 7: (A and B) or (A and C) --> A and (B or C). If there are at least one common predicate in both side, no matter what kind of order is, ie: (A and C) or (B and C), OR (A and C) or (B and A).



Rule 8. A or (A and B) --> A and (1 or B) --> A



Rule 9. A and (A or B) --> A or (1 and B) --> A



5. Description.

This section will describe the algorithm, data structure and implementation of each region. There are four subsections: 5.1 Normalize region, 5.2 CNF region, 5.3 "Simplify AND" region, and 5.4 "Simplify OR" region. Each subsection contains the purpose of the region and when the region to be used. It also contains the class diagram, class description, and pseudocode for each method. In here, these region inherited two superclass: REControlRegion (i.e. CNF region) and RELeafRegion (i.e. "normalize", "Simplify AND", and "Simplify OR" regions). REControlRegion and RELeafRegion are specified in [5]. A control region can combine subregions and choose and branch to suitable subregions to optimize the query. The leaf region, transforms the query itself. For example, the "CNF" region considers the general boolean-expression-query and reorganize it, then send this query branch to "Simplify AND" and "Simplify OR" regions. "Simplify AND" ("Simplify OR") region which are leaf region only consider the cases which the predicates are ANDed (ORed) together, and do not call any child regions while doing the simplification and reordering.

5.1 Normalize region.

- **Purpose of region:**

  Eliminate NOT nodes where possible, keep variables for comparators on left-hand side, where possible. Also, convert obviously trivial comparisons to TRUE and FALSE predicates.

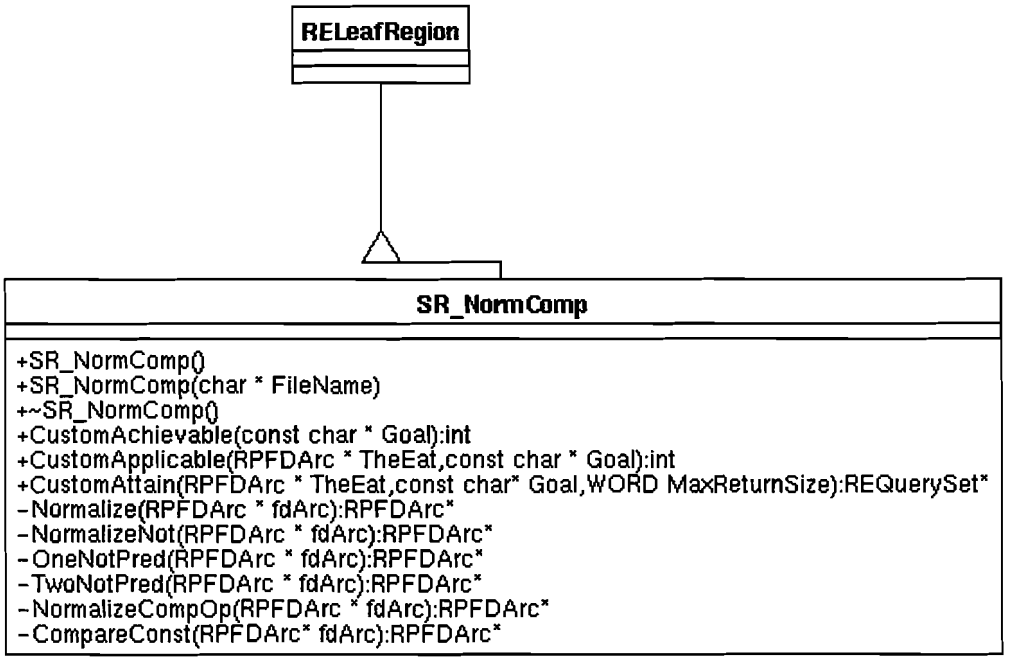
- **Class Description:**

  class SR_NormComp : public RELeafRegion

This is a class for "normalize" region. It is the only region that can attain the "normalize" goal. It provides some methods to normalize the queries. This is the first leaf region and is called for every input query. It tries to eliminate some redundancy, fold NOT into the function it negates, and standardize the predicate. For example: if a variable or path expression is compared to a constant, the constant should be on the left. If there is a "not" over the comparison operator, fold the not into the comparison. i.e.; not (a = b) --> (a != b). If there are two "not"s in a row, just eliminate them, i.e., (not (not (a>b))) --> (a>b).

Figure 5.1.1 is the OMT class diagram of class SR_NormComp.

Figure 5.1.1



I. Public Methods:

SR_NormComp();
Constructor.
SR_NormComp(char * Filename);
This is the normal region constructor and expects the named file to be of the form defined in [7].
~SR_NormComp();
Destructor.
int CustomApplicable (RPFDArc* TheEat, const char * Goal);
This method examines the root function node. CustomApplicable always returns 5, and this region will always be called.
REQuerySet* CustomAttain (RPFDArc* TheEat, const char* Goal, WORD MaxReturnSize);

CustomAttain returns NULL if the query set is not applicable, Otherwise it returns a query set with exactly one query, but that query is normalized. It calls the transformation method (Normalize()) to transform the input EAT and return the REQuerySet of the transformed EAT.

II. Private Methods:

RPFDArc * Normalize(RPFDArc * fdArc);
This method works from the bottom up on the input EAT. First it normalizes the children, and splices in the normalized results. Then it does the following with the current node:
If it is a not node, it normalizes it using NormalizeNot.

RPFDArc * NormalizeNot(RPFDArc * fdArc);
1. If it is a not node over another not node, using TwoNotPred.
2. If it is a not node over a COMPOP node, using OneNotPred.

RPFDArc * OneNotPred(RPFDArc * fdArc);
Changes the sense of the COMPOP and deletes the not node. For example, normalize the query "not x <comparison> y" to "x <opposite comparison> y" exp: not x=y --> x!=y.

RPFDArc * TwoNotPred(RPFDArc * fdArc);
Eliminates two "not" function nodes in a row. exp.: not not x=y --> x=y.

RPFDArc * NormalizeCompOp(RPFDArc * fdArc);
1. Replaces predicates that are obviously TRUE of FALSE. With the true or false predicate, this may cause the whole predicate to evaluate true or false. exp: x=x --> true, x!=x -->false, x<=x -->true.
2. If the expression is compared to constant, put the constant on the right-hand side. When both side are constant, Call CompareConst.

RPFDArc * CompareConst(RPFDArc * fdArc);
Replaces predicates that compare two constants with true or false predicates.

5.2 CNF region.

- **Purpose of region:**

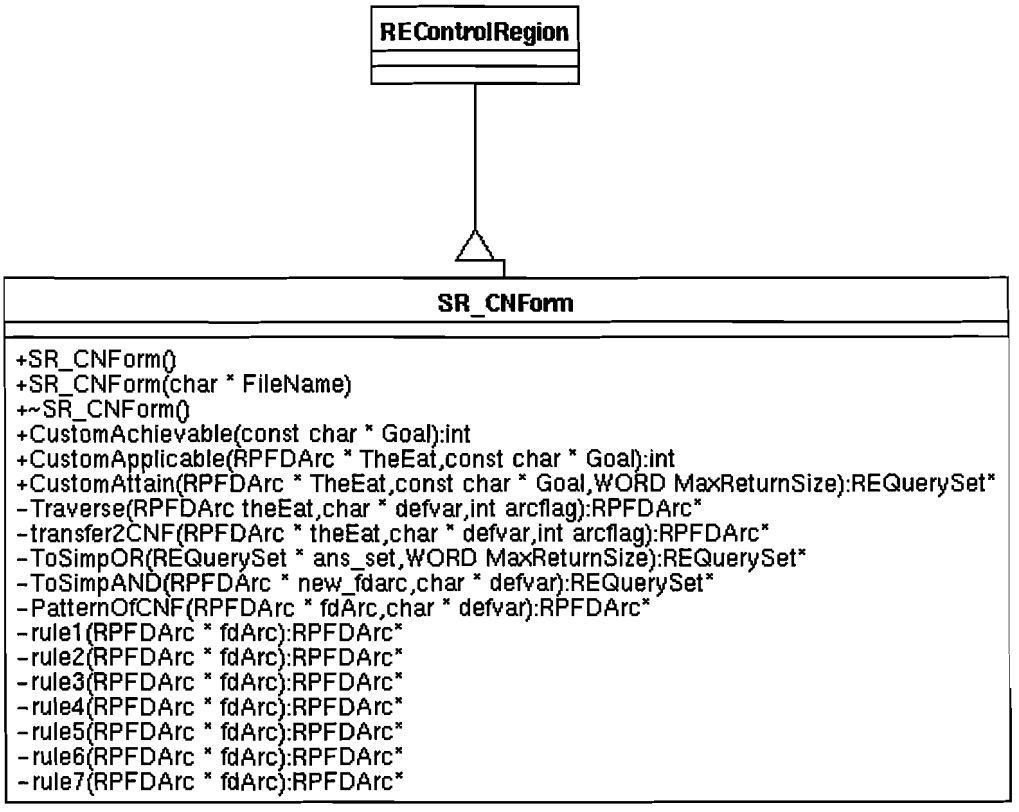
Transform the input query to Conjunctive Normal Form (CNF). Once the predicate is in CNF, for each subpredicate which is the OR of one or more comparisons involving constants, variables, and path expressions, call the "Simplify OR" region to simplify it and reorder it for faster execution. Then call the "Simplify AND" region to simplify the whole predicate and reorder the subpredicates for optimal execution.

- **Class Description:**

class SR_CNForm : public REControlRegion

This class contains methods to transform the input query to Conjunctive Normal Form(CNF) and also branch to "SimplifyAND" and "Simplify OR" regions to simplify and optimize the conjuncts and disjuncts. Right now this region is connected under the "Select" region (see Fig. 2.1). When the root function node is "select", the input query is first optimized, then the predicate is sent into "CNF" region to be transformed and optimized. Fig. 5.2.1 is the OMT class diagram of class SR_CNForm.

Fig. 5.2.1



I. Public Methods:

SR_CNForm();
Constructor.
SR_CNForm(char * FileName);
This is the CNF region construstor and expects the named file to be of the form defined in [7].
~SR_CNForm();
Destructor.
int CustomAchievable (const char * Goal);

CustomAchievable always returns 1. This method makes sure this region is present.

int CustomApplicable (RPFDArc* TheEat, const char * Goal);

CustomApplicable returns 5 if the root function node is "and" or "or", and 0 otherwise.

REQuerySet* CustomAttain (RPFDArc* TheEat, const char* Goal, WORD MaxReturnSize);

CustomAttain returns NULL if the query set is not applicable, Otherwise it returns a query set with exactly one query, but that query is in CNF, and has been simplified and reordered.

II. Private Methods:

RPFDArc * Traverse(RPFDArc * theEat, char * defvar, int arcflag);

Traverse the EAT from the bottom up, calling transfer2CNF at each "or" or "and" node.

RPFDArc * transfer2CNF(RPFDArc * theEat, char * defvar, int arcflag);

1.Check the input RPFDArc, if the current function is an "or" node that may be above some "and" node. If so, call the PatternOfCNF. Otherwise, nothing needs to change, because the EAT is already in CNF.

2. Recursively apply this procedure until all "or" nodes are under the "and" node.

RPFDArc * PatternOfCNF(RPFDArc * fdArc, char * defvar);

Input the RPFDArc and find out the matching pattern and transform to CNF

REQuerySet * ToSimpOR(REQuerySet * and_set, WORD MaxReturnSize);

After the input query transform to CNF, there are some clauses related by ANDs. Each clause consists of one or more comparisons involving constants, variables, and path expressions combined by ORs. So the "Simplify OR" region is called for each clause to simplify the predicates.

REQuerySet * ToSimpAND(RPFDArc * new_fdarc, WORD MaxReturnSize);

After the input query is transformed to CNF, there are some clauses related by ANDs. So after branch to "Simplify OR" region, the new query is still in CNF, is then simplified using the "Simplify AND" region.

RPFDArc * rule1(RPFDArc * fdArc);

((a*b)*c) --> (c*(a*b))

RPFDArc * rule2(RPFDArc * fdArc);

((a+b)+c) --> (c+(a+b))

RPFDArc * rule3(RPFDArc * fdArc, char * defvar);

(a+(c*d)) --> ((a+c)*(a+d))

RPFDArc * rule4(RPFDArc * fdArc, char * defvar);

((c*d)+a) --> (a+(c*d)) --> rule3

RPFDArc * rule5(RPFDArc * fdArc);
((c+d)*a) --> (a*(c+d))

RPFDArc * rule6(RPFDArc * fdArc, char * defvar);
((a*b)+(c*d)) --> ((a+c)*((b+c)*((a+d)*(b+d))))

RPFDArc * rule7(RPFDArc * fdArc, char * defvar);
((a*b)+(a*c)) --> (a*(b+c))

See section 4 for a better description of these rules.

- **Algorithm Description:**

Concept:

For each input query, we try to use some mathematical basis to modify and standardize the input query. Conjunctive Normal Form is a usual method. The following part will describe in detail algorithms for some important methods, and methods that require specialized algorithm.

- **REQuerySet* CustomAttain (RPFDArc* TheEat, const char* Goal, WORD)**

Algorithm:

1. Read the input EAT.
2. Call "Traverse" method to transform the input EAT to CNF(Conjunctive Normal Form).
3. Call "ToSimpOR" method to simplify each subquery.
4. Call "ToSimpAND" method to simplify the new query from step 4.
5. Return new EAT

- **RPFDArc * Traverse(RPFDArc * theEat, char * defvar, int arcflag);**

Algorithm:

```
Traverse(theEAT)
{
    if(not leaf)
       Traverse(left child);
       Traverse(right child);
        transfer2CNF(theEAT);
    else
        transfer2CNF(theEAT);
}
```

- **RPFDArc * transfer2CNF(RPFDArc * theEat, char * defvar, int arcflag);**

  Algorithm:

  1.Check the input EAT,
  2.If this is an "or" node above some "and" nodes, then call PatternOfCNF() method.
  3. Else if the sub-EAT matches the transforming rules(Section4), then call PatternOfCNF() method.
  4. Else do nothing.
  4. Recursively check the whole EAT tree until all the "or" nodes are relocated under all the "and" nodes.
  5. Return the new EAT.

- **RPFDArc * PatternOfCNF(RPFDArc * fdArc, char * defvar);**

  Algorithm:

  <1> Check the input arc --fdArc;
  <2> If the root function node of fdArc is "or";
  <2.1> If the left-child-function node is "or" and right-child-function is not "and" or "or", then call rule2();
  <2.2> If the right-child-function node is "and" and left-child-function is not "and", then call rule3();
  <2.3> If the right-child-function is not "and" and left-child-function is "and", then call rule4();
  <2.4> If both the right-child-function and left-child-function are "and", then
  <2.4.1> If there exists the case match rule 7 of section 4, then call rule7().
  <3> If the root function node of fdArc if "and";
  <3.1> If the left-child-function node is "and" and right-child-function is not "and" or "or", then call rule1();
  <3.2> If the left-child-function node is "or" and right-child-function is not "and" or "or", then call rule5();
  <4> Return the new arc;

  We tested these rules using sixteen test queries. We present the results in section 6.

5.3 "Simplify AND" region.

- **Purpose of region:**

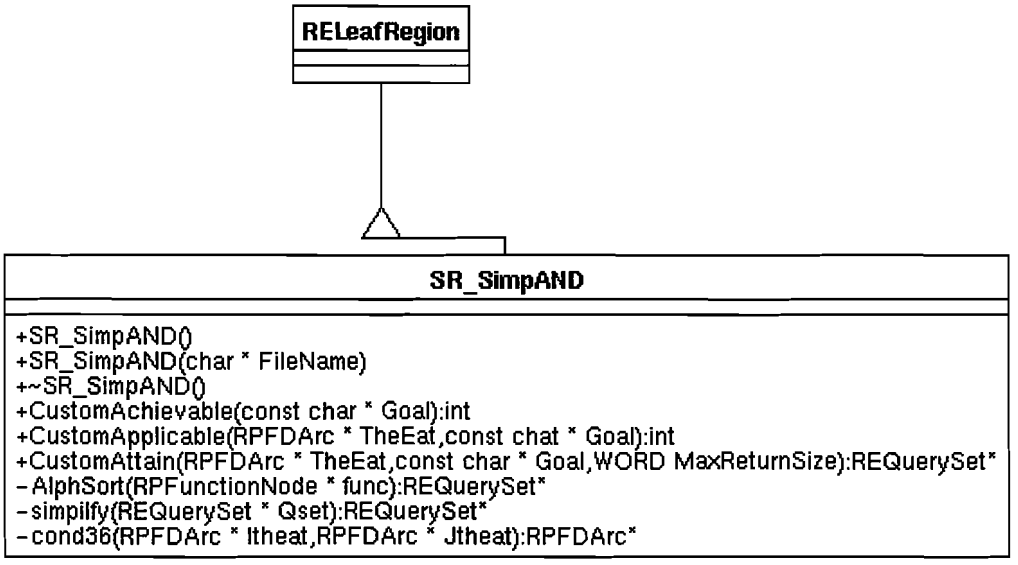
  Optimize predicates with ANDed together. Eliminate redundancy, simplify the predicates algebraically, and sort the predicates into rank order[2].

- **Class Description:**

class SR_SimpAND : public RELeafRegion

This region optimizes predicates that are ANDed together. This region is located under the "CNF" region (see Fig. 2.1). When the root function node is "and", then this region get called and do the transformation. It eliminates redundancy and also simplifies the predicate algebraically. Fig. 5.3.1 is the OMT class diagram of class SR_SimpAND.

Fig. 5.3.1



I. Public Methods:

SR_SimpAND();
Constructor.
SR_SimpAND(char * FileName);
Constructor. The filename must be a valid EPOQ region *.ini file [7].
~SR_SimpAND();
Destructor.
int CustomAchievable (const char * Goal);
CustomAchievable always returns 1.
int CustomApplicable (RPFDArc* TheEat, const char * Goal);
CustomApplicable returns 5 if the root function node is "and", 0 otherwise.
REQuerySet* CustomAttain (RPFDArc* TheEat, const char* Goal, WORD MaxReturnSize);
CustomAttain returns NULL if the query set is not applicable, Otherwise it returns a query set with exactly one query, with the ANDed clauses of the predicate simplified and reordered.

II. Private Methods:

REQuerySet * AlphSort(RPFunctionNode * func);
Store in the REQuerySet, then using Bubble sort to order the REQuerySet by alphabetical order.

REQuerySet * simplify(REQuerySet * Qset);
If there are two predicates which left-hand-side are equal, then there will be 36 conditions to simplify the comparisons. i.e.: X = A and X != B is false, if A=B.

RPFDArc * cond36(RPFDArc * Itheat, RPFDArc* Jtheat);
This tests the two input predicates to see if they satisfy one of the 36 conditions. It returns NULL if there are no changes, or the simplified EAT if the two predicates can be algebraically combined into one. Please refer to the Appendix A for the details of the 36 cases.

- **Algorithm Description:**

This region consider the query like this: (p1 and p2 and p3 and p4 and ...). Each sub query(p1, p2,...) can be: p1=(q1 or q2 or q3 or..), but can contain no "AND" operators. First step, sorts the input predicates(p1, p2, p3,...) in alphabetical order by bubble sort. Secondly, compares each two of them to simplify and eliminate the redundancy. Thirdly, it sorts the remaining predicates by rank. The skeleton of the algorithm is as follows:

INPUT : REQueryset, P1 * P2 * P3 * P4 * ...* Pm
OUTPUT : REQueryset

<1> Sort the input predicates by alphabetical order based on the variable of left-hand side, store them in PSet; name as R[1], R[2], R[3], R[4],..., R[m]
<2> For any two items in R[] set.
<2.1> If the variable of left-hand side of the firstpred and second pred are equal, then,
<2.1.1> Goto and check the 36 situations (refer to Appendix A). if the pair was simplified, then
<2.1.1.1> If we got the true pred, then eliminate it;
<2.1.1.2> Else if we got the false pred. then break and return false;
<2.1.1.3> Else store in the buffer.
<2.1.1.3.1> Uses the new predicate to compare with next item in R[], Go back to step <2.1> and execute it recursively.
<2.1.1.3.2> After above steps, store the new predicate into NPSet.

<2.2> ELSE pick up next two items in R[] set.
<3> Sort NPSet by rank[2];
<4> Return NPSet;

Appendix C contains the test results for the "Simplify AND" region. There are thirty six conditions and some special cases when the right-hand side ia an integer. These results show the algebraically simplified rules. Step 2.2 is based on these rules to simplify the predicate algebraically.

5.4 "Simplify OR" region.

- **Purpose of region:**

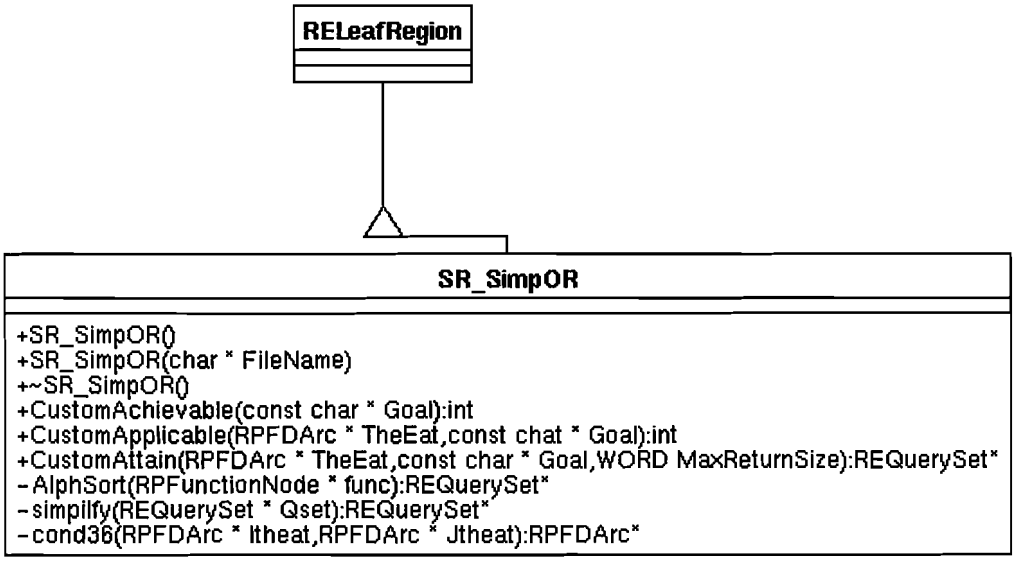
Optimize predicates with ORed together. Eliminate redundancy, simplify the predicates algebraically, and sort the predicates by inverse rank [2].

- **Class Description:**

class SR_SimpOR : public RELeafRegion

This region optimizes predicates that are ORed together. This region are located under the "CNF" region (see Fig. 2.1). When the root function node is "or", then this region is called. It eliminates redundancy and also simplifies the predicate algebraically. Fig. 5.4.1. shows the OMT class diagram of class SR_SimpOR.

Fig. 5.4.1



I. Public Methods:

SR_SimpOR();
Constructor.
SR_SimpOR(char * FileName);
Constructor. The file name must be a valid EPOQ region .ini file.
~SR_SimpOR();
Destructor.
int CustomAchievable (const char * Goal);
CustomAchievable always returns 1.
int CustomApplicable (RPFDArc* TheEat, const char * Goal);
CustomApplicable returns 5 if the root function node is "or", 0 otherwise.
REQuerySet* CustomAttain (RPFDArc* TheEat, const char* Goal, WORD MaxReturnSize);
CustomAttain returns NULL if the query set is not applicable, Otherwise it returns a query set with the "or" clause simplified and reordered.

II. Private Methods:

REQuerySet * AlphSort(RPFunctionNode * func);
Store in the REQuerySet, then using Bubble sort, order the REQuerySet in alphabetical order.

REQuerySet * simplify(REQuerySet * Qset);
Similar to simplify AND.

RPFDArc * cond36(RPFDArc * Itheat, RPFDArc* Jtheat);
This tests the two input predicates to see if they satisfy one of the 36 conditions. It returns NULL if there are no changes, or the simplified EAT if the two predicates can be algebraically combined into one. Please refer to the Appendix B for the details of the 36 cases.

- **Algorithm Description:**

This region considers queries like this: (p1 or p2 or p3 or p4 or ...). First it sorts the input predicates(p1, p2, p3,...) in alphabetic order by bubble sort. Secondly, it compares each two of them to simplify and eliminate the redundancy. Thirdly, it sorts the remaining predicates by inverse rank. The skeleton of the algorithm is as follows:

INPUT : REQueryset, P1 + P2 + P3 + P4 + ...+ Pm
OUTPUT : REQueryset

<1> Sort the input predicates by alphabetical order based on the variable of left-hand side, store them in PSet; name as R[1], R[2], R[3], R[4], ..., R[m]

<2> For any two items in R[] set.
<2.1> If the variable of left-hand side of the firstpred and secondpred are equal, then,
<2.1.1> Goto and check the 36 situations (refer to Appendix B) ,and return simplify result (to be REQuerySet);
<2.1.1.1> If we got the true pred, then break and return true;
<2.1.1.2> Else if we got the false pred. then eliminate it;
<2.1.1.3> Else store the new in the buffer.
<2.1.1.3.1> Uses the new predicate to compare with next item in R[], Go back to step <2.1> and execute it recursively.
<2.1.1.3.2> After above steps, store the new predicate into NPSet.
<2.2> ELSE pick the next two items in R[] set.
<3> Sort NPSet by inverse rank[2];
<4> Return NPSet;

In Appendix C, lists the test results for the "Simplify OR" region. There are thirty six conditions and some special cases when the right-hand side is an integer. These results show the algebraically simplified rules. The step 2.2 is based on these rules to simplify the predicate algebraically.

6. Test Sample.

There are some samples to test our program can work as expecting. The first eight examples are testing the rules of transformation of C.N.F. The other nine samples are testing not only the rules of transforming C.N.F., but also testing the "Simplify OR" and "Simplify AND" region and some tricky cases. After these samples, The test samples of "Simplify OR" and "Simplify AND" are listing in Appendix C. The following samples are in the directory: *"/pro/oodb/epoq/ymc/query/fq*"*. A script with the execution of all these samples is found in Appendix D.

Sample 1:
Input Query:
select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) AND (x.address.number=1910)) (Hotels)
Transforming Procedure:
This query tests the transforming Rule 1. After transforming to C.N.F., the new query is also sent to "Simplify AND" region to be sorted by rank(sorts by decreasing probability of evaluating to false).
Output Query:
select (lambda (x) ((x.name = "ADA") and ((x.address.city.name = "New York") and

(x.address.number = 1910)))) (Hotels)

Sample 2:
Input Query:
select(lambda(x)((x.address.city.name="New York") OR (x.name="ADA")) OR (x.address.number=1910)) (Hotels)
Transforming Procedure:
This query tests the transforming Rule 2. After transforming to C.N.F., the new query is also sent to "Simplify OR" region to be sorted by inverse rank(sorts by increasing probability of evaluating to false).
Output Query:
select (lambda (x) ((x.address.city.name = "New York") or ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Sample 3:
Input Query:
select(lambda(x)(x.address.city.name="New York") OR ((x.name="ADA") AND (x.address.number=1910))) (Hotels)
Transforming Procedure:
This query tests the transforming Rule 3. After transforming to C.N.F., the new query is also sent to "Simplify OR" and "Simplify AND" regions.
Output Query:
select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) and ((x.address.city.name = "New York") or (x.address.number = 1910)))) (Hotels)

Sample 4:
Input Query:
select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR (x.address.number=1910)) (Hotels)
Transforming Procedure:
This query tests the transforming Rule 4. After transforming to C.N.F., the new query is also sent to "Simplify OR" region and "Simplify AND" region.
Output Query:
select (lambda (x) (((x.address.city.name = "New York") or (x.address.number = 1910)) and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Sample 5:
Input Query:
select(lambda(x)((x.address.city.name="New York") OR (x.name="ADA")) AND (x.address.number=1910)) (Hotels)
Transforming Procedure:
This query tests the transforming Rule 5. After transforming to C.N.F., the new query is also sent to "Simplify OR" region and "Simplify AND" region.
Output Query:
select (lambda (x) ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name = "ADA")))) (Hotels)

Sample 6:

Input Query:

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR ((x.address.number=1910) AND (x.stars>=4))) (Hotels)

Transforming Procedure:

This query tests the transforming Rule 6. After transforming to C.N.F., the new query is also sent to "Simplify OR" region and "Simplify AND" region.

Output Query:

select (lambda (x) (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.name = "New York") or (x.address.number = 1910)) and (((x.address.number = 1910) or (x.name = "ADA")) and ((x.address.city.name = "New York") or (x.stars >= 4)))))) (Hotels)

Sample 7:

Input Query:

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR ((x.address.number=1910) AND (x.address.city.name="New York"))) (Hotels)

Transforming Procedure:

This query tests the transforming Rule 7. After transforming to C.N.F., the new query is also sent to "Simplify OR" region and "Simplify AND" region.

Output Query:

select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Sample 8:

Input Query:

select(lambda(x)((x.address.number=5) OR ((x.name="ADA") AND (x.address.number=5)))) (Hotels)

Transforming Procedure:

This query tests the special case of Rule 3. According to the rule, we should get the new query which only contains the common items, in here, is "x.address.number = 5".

Output Query:

select (lambda (x) (x.address.number = 5)) (Hotels)

Sample 9:

Input Query:

select (lambda(x) ((x.stars!=3) AND (x.stars>=3)) AND (x.stars<5)) (Hotels)

Transforming Procedure:

1. Sends the input query into "CNF" region. This query fits the Rule 1, then after transforming, the new query is: "((x.stars<5) AND ((x.stars!=3) AND (x.stars>=3)))".

2. Sends the query into "Simplify AND" region, "((x.stars!=3) AND (x.stars>=3))" will become "(x.stars>3)". Then "(x.stars<5) AND (x.stars>3)" will be "(x.stars=4)".

Output Query:

select (lambda (x) (x.stars = 4)) (Hotels)

Sample 10:
Input Query:
select (lambda(x) ((x.stars!=3) OR (x.stars>=3)) OR (x.stars<5)) (Hotels)
Transforming Procedure:
1. Sends the input query into "CNF" region. This query fits the Rule 2, then after transforming, the new query is: "((x.stars<5) OR ((x.stars!=3) OR (x.stars>=3)))".
2. Sends the query into "Simplify OR" region, "((x.stars!=3) OR (x.stars>=3))" will become "true". Then "(x.stars<5) AND true" will be "true".
Output Query:
select (lambda (x) true) (Hotels)

Sample 11:
InputQuery:
select (lambda(x) (x.stars!=3) OR ((x.stars>=3) AND (x.stars<5))) (Hotels)
Transforming Procedure:
1. Sends the input query into "CNF" region. This query fits the Rule 3, so after transforming, the new query is: "(((x.stars!=3) OR (x.stars>=3)) AND ((x.stars !=3) OR (x.stars<5)))".
2. Sends the transformed query into "Simplify OR" region, "((x.stars!=3) OR (x.stars>=3))" will become "true". "((x.stars!=3) OR (x.stars<5))" will be "true". So after leaving "Simplify OR" region, we get the new query "true".
3. Checks the new query and notes that we don't need to send it to "Simplify AND" region.
Output Query:
select (lambda (x) true) (Hotels)

Sample 12:
Input Query:
select (lambda(x) ((x.stars!=3) AND (x.stars>=3)) OR (x.stars<5)) (Hotels)
Transforming Procedure:
1. Sends the input query into "CNF" region. This query fits the Rule 4, so after transforming, the new query is: "(((x.stars<5) OR (x.stars!=3)) AND ((x.stars <5) OR (x.stars>=3)))".
2. Sends the new query into "Simplify OR" region, "((x.stars!<5) OR (x.stars!=3))" will become "true". "((x.stars<5) OR (x.stars>=3))" will be "true". So after leaving "Simplify OR" region, we get the new query "true".
3. Checks this new query and note that we don't need to send it to "Simplify AND" region.
Output Query:
select (lambda (x) true) (Hotels)

Sample 13:
Input Query:
select (lambda(x) ((x.stars!=2) OR (x.stars>=3)) AND (x.stars<2)) (Hotels)
Transforming Procedure:
1. Sending the input query into "CNF" region. This query fits the Rule 5, so after trans-

forming, the new query is: "((x.stars<2) AND ((x.stars!=2) OR (x.stars>=3)))".

2. Sending the query into "Simplify OR" region, "((x.stars!=2) OR (x.stars>=3))" will become "(x.stars!=2)". Then "((x.stars<2) AND (x.stars!=2))" will be "(x.stars<2)".So after leaving "Simplify OR" region, we get the new query is "(x.stars<2)".

3. Sends the new query to the "Simplify AND" region. Since there is only one predicate, the "Simplify AND" region will do nothing.

Output Query:

select (lambda (x) (x.stars < 2)) (Hotels)

Sample 14:

Input Query:

select (lambda(x) ((x.stars!=2) AND (x.stars<3)) OR ((x.stars!=3) AND (x.stars<4)) OR ((x.stars<=5) AND (x.stars>=2))) (Hotels)

Transforming Procedure:

1. This input query is D.N.F. (Disjunctive Normal Form). After sending into "CNF" region, We got very huge query(24 predicates). Let's just analyze the D.N.F predicate to check that we got the right out put, no matter you use C.N.F. or D.N.F. to standardize the input query.

1a. "((x.stars!=2) AND (x.stars<3))" will become "(x.stars<2)". "((x.stars!=3) AND (x.stars<4))" will become "(x.stars<3)". "((x.stars<=5) AND (x.stars>=2))" will be the same.

1b. So we got the new query:"(x.stars<2) OR (x.stars<3) OR (x.stars<=5 AND x.stars>=2)". This new query can also be simplified as "(x.stars<3) OR (x.stars<=5 AND x.stars>=2)". For further simplifying, it will be "(x.stars<=5)".

Output Query:

select (lambda (x) (x.stars <= 5)) (Hotels)

Sample 15:

Input Query:

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR ((x.address.city.name="New York") AND (x.address.number=1910)) OR ((x.address.city.name="New York") AND (x.name="ADA"))) (Hotels)

Transforming Procedure:

1. In the "CNF" region, "((x.address.city.name="New York") AND (x.name="ADA")) OR ((x.address.city.name="New York") AND (x.address.number=1910))" fits the Rule 7.

2. So according to the transformation procedure, it will become as "((x.address.city.name="New York") AND ((x.name="ADA") OR (x.address.number=1910)))". Then the query becomes "(((x.address.city.name="New York") AND ((x.name="ADA") OR (x.address.number=1910))) OR(((x.address.city.name="New York") AND (x.name="ADA"))).

3. The transformed query also fits the Rule 7. Then do it again. The new query is "(((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name = "ADA")))). The duplicate predicate "(x.name="ADA")" is eliminated.

5. Sending the new query to "Simplify OR" and "Simplify AND" regions.

Output Query:

select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Sample 16:
Input Query:
select(lambda(x)((((x.address.city.name="New York") AND false) OR (x.name="ADA") OR ((x.stars=4) AND (x.address.number=1910))))) (Hotels)
Transforming Procedure:
1. In "CNF" region, obviously "((x.address.city.name="New York") AND false)" will be "false" in the CNF region (the CNF region will check the special cases when the left-leaf is true or false). Then "(false OR (x.name="ADA"))"will be "(x.name="ADA")" in the CNF region. After organizing, the new query is "((x.name="ADA") OR ((x.stars=4) AND (x.address.number=1910))). This query fits Rule 3. Then transform it.
2. Sending the new query to "Simplify OR" and "Simplify AND" regions.
Output Query:
select (lambda (x) (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

7. Future Work.

In our implementation, the strategy of transformation the input query to C.N.F. is working well. Almost every case is considered. We also attain the optimized query after executing our procedures(please see section 3.). Some input queries(for example, sample 1 to simple 8) are reordered by their cost(also known as rank), and some of them(for example, sample 9 to sample 14) are simplified by more precise expression. Everything seems to be prefect. But we also find some problems need to be considered in the future.

Remember the sample 14, in section 3. Let's simplify the expression of that query as: (A*B)+(C*D)+(E*F), in here "*" means "and", "+" means "or", just like the general logical textbook. This is D.N.F. (Disjunctive Normal Form). When we got this input query and try to transform it to C.N.F. We will get eight items(2*2*2=8) in our C.N.F. expression. As our experiment, since we use postorder to sort the EAT tree, it won't take much time to transform to C.N.F. The most expensive thing is to send each item to "Simplify OR" and "Simplify AND" regions. If we have "DNF" region, we only need to send three items((A*B), (C*D) ,and (E*F)) to "Simplify AND" region and one to "Simplify OR" region. But if we transform it to C.N.F., there are eight items sending to "Simplify OR" region and one to "Simplify AND" region. It will takes some time to send and receive the data between regions. On the other hand, if the input query is: (A+B)*(C+D)*(E+F) and assuming we had "DNF" region, we got the similar problem. So we may want to find some strategy to figure out the input query fits D.N.F. or C.N.F. transformation. This is complicated when a query combines these two cases,
(A*B)+(C*D)+(E*F)+(G+H)*(I+J)*(K+L).

Considering the other question. Let's back to our example--sample 14. If we have the "DNF" region, we don't need to transform if we send this query into the region. But after sending to "Simplify AND" region and "Simplify OR" region, we get the new query is:(x.stars<3) OR (x.stars<=5 AND x.stars>=2) and we can't simplify any more. But if we input this query into our test module, we can get the simplest expression: (x.stars <= 5).

Since D.N.F. and C.N.F. are complementary, if there are some questions in one normal form, another must have the same problems. These are tricky question and need to be consider in the future.

[References]

[1] Theodore W. Leung, Gail Mitchell, Bharathi Subramanian, Bennet Vance, Scott L. Vandenberg, ans Stanley B. Zdonik. The AQUA data model and algebra. Technical Report CS-93-09, Brown University Department of Computer Science, March 1993.

[2] Joseph M. Hellerstein, Practical Predicate Placement, SIGMOD 94-5/94 Minneapolis, Minnesota, USA.

[3] Epp, Susanna S., Discrete mathematics with applications, Belmont, Calif. : Wadsworth Pub. Co., c1990.

[4] Gail Mitchell, Stanley B. Zdonik, and Umeshwar Dayal. An Architecture for Query Processing in Persistent Object Stores.. In Proc. of the 25th Hawaii Int'l Conf. on System Sciences, Jan. 1991.

[5] Marian H. Nodine, Farah B, Abbas, and Mitch Cherniack. The EPOQ Query Optimizer for Objext-Oriented Databases Design Specification. Brown University, Oct., 1994

[6] Thomas H. Corman, Charles E. Leiserson, and Ronald L.Rivest, Introduction to Algorithms, The MIT press, Cambridge, Massachusetts, McGraw-Hill Book Company. 1994

[7] Laurent Hasson, EPOQ Regions New Design Specifications, Brown University, August 23, 1994.

Appendix A.

The following lists the detailed conditions used in the "Simplify AND" region to simplify the query. There are 36 conditions and some special cases when the right hand sides are integers.

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
1	X=A X=B	A=B A!=B	X=A FALSE
2	X=A X!=B	A=B A!=B	X=A FALSE
3	X=A X>=B	A>=B A<B	X=A FALSE
4	X=A X<=B	A<=B A>B	X=A FALSE
5	X=A X>B	A>B A<=B	X=A FALSE
6	X=A X<B	A<B A>=B	X=A FALSE
7	X!=A X=B	A=B A!=B	FALSE X=B
8	X!=A X!=B	A=B A!=B	X!=A THE SAME
9	X!=A X>=B	A=B A>B A<B	X>B THE SAME X>=B
10	X!=A X<=B	A=B A>B A<B	X<B X<=B THE SAME
11	X!=A X>B	A<=B A>B	X>B THE SAME
11 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X>A THE SAME
12	X!=A X<B	A>=B A<B	X<B THE SAME

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
12 SPECIAL CASE	A, B ARE INTEGER	A<B AND A+1=B ELSE	X<A THE SAME
13	X>=A X=B	A<=B A>B	X=B FALSE
14	X>=A X!=B	A=B A>B A<B	X>A X>=A THE SAME
15	X>=A X>=B	A>=B A<B	X>=A X>=B
16	X>=A X<=B	A=B A<B A>B	X=A THE SAME FALSE
17	X>=A X>B	A>B A<=B	X>=A X>B
18	X>=A X<B	A>=B A<B	FALSE THE SAME
18 SPECIAL CASE	A, B ARE INTEGER	A<B AND A+1=B ELSE	X=A THE SAME
19	X<=A X=B	A>=B A<B	X=B FALSE
20	X<=A X!=B	A=B A>B A<B	X<A THE SAME X>=A
21	X<=A X>=B	A=B A<B A>B	X=B FALSE THE SAME
22	X<=A X<=B	A<=B A>B	X<=A X<=B
23	X<=A X>B	A<=B A>B	FALSE THE SAME

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
23 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X=A THE SAME
24	X<=A X<B	A>=B A<B	X<B X<=A
25	X>A X=B	A<B A>=B	X=B FALSE
26	X>A X!=B	A>=B A<B	X>A THE SAME
26 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X>B THE SAME
27	X>A X>=B	A<B A>=B	X>=B X>A
28	X>A X<=B	A>=B A<B	FALSE THE SAME
28 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X=A THE SAME
29	X>A X>B	A>=B A<B	X>A X>B
30	X>A X<B	A>=B A<B	FALSE THE SAME
30 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B A+2=B ELSE	FALSE X=A THE SAME
31	X<A X=B	A>B A<=B	X=B FALSE
32	X<A X!=B	A<=B A>B	X<A THE SAME
32 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X<B THE SAME

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
33	X<A X>=B	A<=B A>B	FALSE THE SAME
33 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X=B THE SAME
34	X<A X<=B	A<=B A>B	X<A X<=B
35	X<A X>B	A<=B A>B	FALSE THE SAME
35 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 A=B+2 ELSE	FALSE X=A THE SAME
36	X<A X<B	A<=B A>B	X<A X<B

Appendix B.

The following lists the detailed conditions used in the "Simplify OR" region to simplify the query. There are 36 conditions and some special cases when the right hand sides are integers.

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
1	X=A X=B	A=B A!=B	X=A THE SAME
2	X=A X!=B	A=B A!=B	TRUE X!=B
3	X=A X>=B	A>=B A<B	X>=B THE SAME
3 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X>=A THE SAME
4	X=A X<=B	A<=B A>B	X<=B THE SAME
4 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X<=A THE SAME
5	X=A X>B	A=B A>B A<B	X>=B X>B THE SAME
6	X=A X<B	A=B A<B A>B	X<=B X<B THE SAME
7	X!=A X=B	A=B A!=B	TRUE X!=A
8	X!=A X!=B	A=B A!=B	X!=A THE SAME
9	X!=A X>=B	A>=B A<B	TRUE X!=A
10	X!=A X<=B	A>B A<=B	X!=A TRUE
11	X!=A X>B	A<=B A>B	X!=A TRUE

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
12	X!=A X<B	A>=B A<B	X!=A TRUE
13	X>=A X=B	A<=B A>B	X>=A THE SAME
13 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X>=B THE SAME
14	X>=A X!=B	A>B A<=B	X!=B TRUE
15	X>=A X>=B	A<=B A>B	X>=A X>=B
16	X>=A X<=B	A<=B A>B	TRUE THE SAME
16 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	TRUE THE SAME
17	X>=A X>B	A<=B A>B	X>=A X>B
18	X>=A X<B	A<=B A>B	TRUE THE SAME
18 SPECIAL CASE	A, B ARE INTEGER	A>B AND A=B+1 ELSE	X!=B THE SAME
19	X<=A X=B	A>=B A<B	X<=A THE SAME
19 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X<=B THE SAME
20	X<=A X!=B	A>=B A<B	TRUE X!=B
21	X<=A X>=B	A>=B A<B	TRUE THE SAME

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
21 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	TRUE THE SAME
22	X<=A X<=B	A>=B A<B	X<=A X<=B
23	X<=A X>B	A>=B A<B	TRUE THE SAME
23 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X!=B THE SAME
24	X<=A X<B	A>=B A<B	X<=A X<B
25	X>A X=B	A=B A<B A>B	X>=A X>A THE SAME
26	X>A X!=B	A>=B A<B	X!=B TRUE
27	X>A X>=B	A<B A>=B	X>A X>=B
28	X>A X<=B	A<=B A>B	TRUE THE SAME
28 SPECIAL CASE	A,B ARE INTEGER	A>B AND A=B+1 ELSE	X!=A THE SAME
29	X>A X>B	A<=B A>B	X>A X>B
30	X>A X<B	A=B A>B A<B	X!=A THE SAME TRUE
31	X<A X=B	A=B A>B A<B	X<=A X<A THE SAME
32	X<A X!=B	A<=B A>B	X!=B TRUE

Table 1:

Condition #	Input queries	Comparing R.H.S.	Result
33	X<A X>=B	A>=B A<B	TRUE THE SAME
33 SPECIAL CASE	A,B ARE INTEGER	A<B AND A+1=B ELSE	X!=A THE SAME
34	X<A X<=B	A>B A<=B	X<A X<=B
35	X<A X>B	A=B A<B A>B	X!=A THE SAME TRUE
36	X<A X<B	A>=B A<B	X<A X<B

Appendix C.

A. The test results of "Simplify AND" region. Note that the variable of left hand side must be the same. "org" means the input query. The next line following this one is the output.

```
org x=4 and x=4
x=4
org x=4 and x=5
false
org x=4 and x!=4
false
org x=4 and x!=5
x=4
org x=4 and x>=3
x=4
org x=4 and x>=5
false
org x=4 and x<=5
x=4
org x=4 and x<=3
false
org x=4 and x>3
x=4
org x=4 and x>4
false
org x=4 and x<5
x=4
org x=4 and x<4
false
org x!=4 and x=4
false
org x!=4 and x=3
x=3
org x!=4 and x!=4
x!=4
org x!=4 and x!=3
x!=4 and x!=3
org x!=4 and x>=4
x > 4
org x!=4 and x>=3
x!=4 and x>=3
org x!=4 and x>=5
x>=5
org x!=4 and x<=4
x < 4
org x!=4 and x<=3
x<=3
org x!=4 and x<=5
x!=4 and x<=5
org x!=4 and x>3
x > 4
org x!=4.0 and x>3.0
x!=4.0 and x>3.0
org x!=4 and x>5
x>5
org x!=4 and x<3
x<3
org x!=4 and x<5
x < 4
org x!=4.0 and x<5.0
x!=4.0 and x<5.0
org x>=4 and x=3
false
org x>=4 and x=5
x=5
org x>=4 and x!=4
x > 4
org x>=4 and x!=3
x>=4
org x>=4 and x!=5
x>=4 and x!=5
org x>=4 and x>=3
x>=4
org x>=4 and x<=4
x = 4
org x>=4 and x<=3
false
org x>=4 and x<=5
x>=4 and x<=5
org x>=4 and x>3
x>=4
org x>=4 and x>5
x>5
org x>=4 and x<3
false
org x>=4 and x<5
x = 4
org x>=4.0 and x<5.0
x>=4.0 and x<5.0
org x>=4 and x<6
x>=4 and x<6
```

org x<=4 and x=3
x=3
org x<=4 and x=5
false
org x<=4 and x!=4
x < 4
org x<=4 and x!=3
x<=4 and x!=3
org x<=4 and x!=5
x<=4
org x<=4 and x>=4
x = 4
org x<=4 and x>=3
x<=4 and x>=3
org x<=4 and x>=5
false
org x<=4 and x<=3
x<=3
org x<=4 and x<=5
x<=4
org x<=4 and x>5
false
org x<=4 and x>3
x = 4
org x<=4.0 and x>3.0
x<=4.0 and x>3.0
org x<=4 and x>2
x<=4 and x>2
org x<=4 and x<3
x<3
org x<=4 and x<5
x<=4
org x>4 and x=5
x=5
org x>4 and x=3
false
org x>4 and x!=3
x>4
org x>4 and x!=5
x > 5
org x>4.0 and x!=5.0
x>4.0 and x!=5.0
org x>4 and x>=5
x>=5
org x>4 and x>=3
x>4
org x>4 and x<=3
false
org x>4 and x<=5
x = 5
org x>4.0 and x<=5.0
x>4.0 and x<=5.0
org x>4 and x<=6
x>4 and x<=6
org x>4 and x>3
x>4
org x>4 and x>5
x>5
org x>4 and x<3
false
org x>4 and x<5
false
org x>4.0 and x<5.0
x>4.0 and x<5.0
org x>4 and x<6
x = 5
org x>4.0 and x<6.0
x>4.0 and x<6.0
org x>4 and x<7
x>4 and x<7
org x<4 and x=3
x=3
org x<4 and x=5
false
org x<4 and x!=3
x < 3
org x<4.0 and x!=3.0
x<4.0 and x!=3.0
org x<4 and x!=5
x<4
org x<4 and x>=5
false
org x<4 and x>=3
x = 3
org x<4.0 and x>=3.0
x<4.0 and x>=3.0
org x<4 and x>=2
x<4 and x>=2
org x<4 and x>5
false
org x<4 and x>3
false

org x<4.0 and x>3.0
x<4.0 and x>3.0
org x<4 and x>2
x = 3
org x<4.0 and x>2.0
x<4.0 and x>2.0
org x<4 and x>1
x<4 and x>1
org x<4 and x<5
x<4
org x<4 and x<3
x<3

B. The test results of the "Simplify OR" region. Note that the variable on the left hand side must be the same. "org" means the input query. The next line following this one is the output.

org x=4 or x=4
x=4
org x=4 or x=3
x=4 or x=3
org x=4 or x!=4
true
org x=4 or x!=3
x!=3
org x=4 or x>=3
x>=3
org x=4 or x>=5
x >= 4
org x=4.0 or x>=5.0
x=4.0 or x>=5.0
org x=4 or x>=6
x=4 or x>=6
org x=4 or x<=5
x<=5
org x=4 or x<=3
x <= 4
org x=4.0 or x<=3.0
x=4.0 or x<=3.0
org x=4 or x<=2
x=4 or x<=2
org x=4 or x>4
x >= 4
org x=4 or x>3
x>3
org x=4 or x>5
x=4 or x>5
org x=4 or x<4
x <= 4
org x=4 or x<5
x<5
org x=4 or x<3
x=4 or x<3
org x!=4 or x=4
true
org x!=4 or x=3
x!=4
org x!=4 or x!=4
x!=4
org x!=4 or x!=3
x!=4 or x!=3
org x!=4 or x>=3
true
org x!=4 or x>=5
x!=4
org x!=4 or x<=5
true
org x!=4 or x<=3
x!=4
org x!=4 or x>5
x!=4
org x!=4 or x>3
true
org x!=4 or x<3
x!=4
org x!=4 or x<5
true
org x>=4 or x=5
x>=4
org x>=4 or x=3
x >= 3
org x>=4.0 or x=3.0
x>=4.0 or x=3.0
org x>=4 or x=2
x>=4 or x=2
org x>=4 or x!=5
true
org x>=4 or x!=3
x!=3
org x>=4 or x>=5

x>=4
org x>=4 or x>=3
x>=3
org x>=4 or x<=5
true
org x>=4 or x<=3
true
org x>=4.0 or x<=3.0
x>=4.0 or x<=3.0
org x>=4 or x<=2
x>=4 or x<=2
org x>=4 or x>3
x>3
org x>=4 or x>5
x>=4
org x>=4 or x<5
true
org x>=4.0 or x<5.0
true
org x>=4 or x<3
x != 3
org x>=4.0 or x<3.0
x>=4.0 or x<3.0
org x>=4 or x<2
x>=4 or x<2
org x<=4 or x=3
x<=4
org x<=4 or x=5
x <= 5
org x<=4.0 or x=5.0
x<=4.0 or x=5.0
org x<=4 or x=6
x<=4 or x=6
org x<=4 or x!=3
true
org x<=4 or x!=5
x!=5
org x<=4 or x>=3
true
org x<=4 or x>=5
true
org x<=4.0 or x>=5.0
x<=4.0 or x>=5.0
org x<=4 or x>=6
x<=4 or x>=6
org x<=4 or x<=3
x<=4
org x<=4 or x<=5
x<=5
org x<=4 or x>3
true
org x<=4 or x>5
x != 5
org x<=4.0 or x>5.0
x<=4.0 or x>5.0
org x<=4 or x>6
x<=4 or x>6
org x<=4 or x<3
x<=4
org x<=4 or x<5
x<5
org x>4 or x=4
x >= 4
org x>4 or x=5
x>4
org x>4 or x=3
x>4 or x=3
org x>4 or x!=3
x!=3
org x>4 or x!=5
true
org x>4 or x>=5
x>4
org x>4 or x>=3
x>=3
org x>4 or x<=5
true
org x>4 or x<=3
x != 4
org x>4.0 or x<=3.0
x>4.0 or x<=3.0
org x>4 or x<=2
x>4 or x<=2
org x>4 or x>5
x>4
org x>4 or x>3
x>3
org x>4 or x<4
x != 4
org x>4 or x<5
true
org x>4 or x<3

x>4 or x<3
org x<4 or x=4
x <= 4
org x<4 or x=3
x<4
org x<4 or x=5
x<4 or x=5
org x<4 or x!=5
x!=5
org x<4 or x!=3
true
org x<4 or x>=3
true
org x<4 or x>=5
x != 4
org x<4.0 or x>=5.0
x<4.0 or x>=5.0
org x<4 or x>=6
x<4 or x>=6
org x<4 or x<=5
x<=5
org x<4 or x<=3
x<4
org x<4 or x>4
x != 4
org x<4 or x>3
true
org x<4 or x>5
x<4 or x>5
org x<4 or x<3
x<4
org x<4 or x<5
x<5

Appendix D. A script of the execution.

I. CNF region

```
Script started on Wed May 08 12:14:57 1996
fiddle:/u/ymc% epoq
fiddle:oodb/epoq/ymc% p

************************************************************
*****************   G r O O V Y   E P O Q   *****************
************************************************************
*** Region 'Top' construction succeeded ***
*** Region 'Planner' construction succeeded ***
*** Region 'Optimizer' construction succeeded ***
*** Region 'Select' construction succeeded ***
*** Region 'Apply' construction succeeded ***
*** Region 'Normalize' construction succeeded ***
*** Region 'SimpAND' construction succeeded ***
*** Region 'Simplify OR' construction succeeded ***
*** Region 'CNF' construction succeeded ***
Successful plugging of NormComp on Top
Successful plugging of Optimizer on Top
Successful plugging of Select on Optimizer
Successful plugging of Apply on Optimizer
Successful plugging of CNForm on Select
Successful plugging of SimpAND on CNF
Successful plugging of SimpOR on CNF
0: Top
    |-- 0: Normalize
    |-- 1: Optimizer
         |-- 0: Select
         |    |-- 0: CNF
         |         |-- 0: SimpAND
         |         |-- 1: Simplify OR
         |-- 1: Apply
Added Globals in file /pro/oodb/epoq/ymc/SM/TravelSchema successfully.
Added Tuples in file /pro/oodb/epoq/ymc/SM/TravelTuples successfully.
Added derived types in file /pro/oodb/epoq/ymc/SM/DerivedTypes successfully.
Added global symbols in file /pro/oodb/epoq/ymc/SM/TravelGlobals successfully.

//#### Query 1 #####//

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) AND (x.address.num
ber=1910)) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.ad
dress.number = 1910))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.ad
dress.number = 1910))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) a
nd (x.address.number = 1910))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.ad
dress.number = 1910))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.add
ress.number = 1910))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.address.number = 1910)
)

SR_CNForm customattain
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) and (x.address.number =
 1910))
Output: ((x.address.number = 1910) and ((x.address.city.name = "New York") and (x.name =
"ADA")))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.number = 1910)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.city.name = "New York")

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.name = "ADA")

Region Simplify OR Goal or.  No return queries.

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.address.number = 1910) and ((x.address.city.name = "New York") and (x.name = "ADA"))
)

SR_simpAND customattain
query 0: (x.name = "ADA")
query 1: (x.address.city.name = "New York")
query 2: (x.address.number = 1910)
Region SimpAND Goal and.  Return queries are:
query 0: ((x.name = "ADA") and ((x.address.city.name = "New York") and (x.address.number
= 1910)))

Region CNF Goal CNF.  Return queries are:
query 0: ((x.name = "ADA") and ((x.address.city.name = "New York") and (x.address.number
= 1910)))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) ((x.name = "ADA") and ((x.address.city.name = "New York") and
 (x.address.number = 1910)))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.name = "ADA") and ((x.address.city.name = "New York") and
 (x.address.number = 1910)))) (Hotels)
```

```
Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.name = "ADA") and ((x.address.city.name = "New York") an
d (x.address.number = 1910)))) (Hotels)


//#### Query 2 ####//

select(lambda(x)((x.address.city.name="New York") OR (x.name="ADA")) OR (x.address.numbe
r=1910)) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) or
 (x.address.number = 1910))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.address.number = 1910))

SR_CNForm customattain
entring rule2 -- (a+b)+c --> c+(a+b)
Input: (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.address.number =
1910))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.address.city.name = "New York") or (x.name = "ADA")) or (x.address.number =
1910))
Output: ((x.address.number = 1910) or ((x.address.city.name = "New York") or (x.name = "
ADA")))
Output: ((x.address.number = 1910) or ((x.address.city.name = "New York") or (x.name = "
ADA")))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.number = 1910) or ((x.address.city.name = "New York") or (x.name = "ADA")))

query 0: (x.address.city.name = "New York")
query 1: (x.address.number = 1910)
query 2: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or ((x.address.number = 1910) or (x.name = "
ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: ((x.address.city.name = "New York") or ((x.address.number = 1910) or (x.name = "
ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") or ((x.address.number = 1
910) or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") or ((x.address.number = 1
910) or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") or ((x.address.number = 1
910) or (x.name = "ADA")))) (Hotels)


//#### Query 3 ####//

select(lambda(x)(x.address.city.name="New York") OR ((x.name="ADA") AND (x.address.number
=1910))) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.addre
ss.number = 1910)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.addre
ss.number = 1910)))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") or ((x.name = "ADA") and
(x.address.number = 1910)))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.addre
ss.number = 1910)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.addre
ss.number = 1910)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.address.number = 1910)))

SR_CNForm customattain
```

```
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.address.city.name = "New York") or ((x.name = "ADA") and (x.address.number =
1910)))
Output: (((x.address.city.name = "New York") or (x.name = "ADA")) and ((x.address.city.n
ame = "New York") or (x.address.number = 1910)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.city.name = "New York") or (x.name = "ADA"))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.name = "ADA"))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.city.name = "New York") or (x.address.number = 1910))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.address.number = 1910)
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.address.number = 1910))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 (((x.address.city.name = "New York") or (x.name = "ADA")) and ((x.address.city.name = "
New York") or (x.address.number = 1910)))

SR_simpAND customattain
query 0: ((x.address.city.name = "New York") or (x.name = "ADA"))
query 1: ((x.address.city.name = "New York") or (x.address.number = 1910))
Region SimpAND Goal and.  Return queries are:
query 0: (((x.address.city.name = "New York") or (x.name = "ADA")) and ((x.address.city.
name = "New York") or (x.address.number = 1910)))

Region CNF Goal CNF.  Return queries are:
query 0: (((x.address.city.name = "New York") or (x.name = "ADA")) and ((x.address.city.
name = "New York") or (x.address.number = 1910)))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) an
d ((x.address.city.name = "New York") or (x.address.number = 1910)))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) an
d ((x.address.city.name = "New York") or (x.address.number = 1910)))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) an
d ((x.address.city.name = "New York") or (x.address.number = 1910)))) (Hotels)


//#### Query 4 ####//

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR (x.address.numb
er=1910)) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.add
ress.number = 1910))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.add
ress.number = 1910))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or
 (x.address.number = 1910))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.addr
ess.number = 1910))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.address.number = 1910))

SR_CNForm customattain
entring rule4 -- (c*d)+a --> a+(c*d) --> rule3
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.address.number =
1910))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) or (x.address.number =
1910))
Output: ((x.address.number = 1910) or ((x.address.city.name = "New York") and (x.name = "
ADA")))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.address.number = 1910) or ((x.address.city.name = "New York") and (x.name = "A
DA")))
Output: (((x.address.number = 1910) or (x.address.city.name = "New York")) and ((x.addres
s.number = 1910) or (x.name = "ADA")))
Output: (((x.address.number = 1910) or (x.address.city.name = "New York")) and ((x.addres
s.number = 1910) or (x.name = "ADA")))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.number = 1910) or (x.address.city.name = "New York"))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.address.number = 1910)
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.address.number = 1910))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.number = 1910) or (x.name = "ADA"))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.number = 1910)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.number = 1910) or (x.name = "ADA"))
```

```
Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 (((x.address.city.name = "New York") or (x.address.number = 1910)) and ((x.address.numb
er = 1910) or (x.name = "ADA")))

SR_simpAND customattain
query 0: ((x.address.city.name = "New York") or (x.address.number = 1910))
query 1: ((x.address.number = 1910) or (x.name = "ADA"))
Region SimpAND Goal and.  Return queries are:
query 0: (((x.address.city.name = "New York") or (x.address.number = 1910)) and ((x.addr
ess.number = 1910) or (x.name = "ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: (((x.address.city.name = "New York") or (x.address.number = 1910)) and ((x.addr
ess.number = 1910) or (x.name = "ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.address.number =
1910)) and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.address.number =
1910)) and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.address.number =
1910)) and ((x.address.number = 1910) or (x.name = "ADA")))) (Hotels)


//#### Query 5 ####//

select(lambda(x)((x.address.city.name="New York") OR (x.name="ADA")) AND (x.address.numb
er=1910)) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.add
ress.number = 1910))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.add
ress.number = 1910))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) an
d (x.address.number = 1910))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.add
ress.number = 1910))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.add
ress.number = 1910))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.address.number = 1910))

SR_CNForm customattain
entring rule5 -- (c+d)*a --> a*(c+d)
Input: (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.address.number =
1910))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.address.city.name = "New York") or (x.name = "ADA")) and (x.address.number =
1910))
Output: ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name = "
ADA")))
Output: ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name = "
ADA")))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.number = 1910)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.city.name = "New York") or (x.name = "ADA"))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.name = "ADA"))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name = "ADA")))

SR_simpAND customattain
query 0: (x.address.number = 1910)
query 1: ((x.address.city.name = "New York") or (x.name = "ADA"))
Region SimpAND Goal and.  Return queries are:
query 0: ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name =
"ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: ((x.address.number = 1910) and ((x.address.city.name = "New York") or (x.name =
"ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) ((x.address.number = 1910) and ((x.address.city.name = "New Y
ork") or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.number = 1910) and ((x.address.city.name = "New Y
ork") or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.number = 1910) and ((x.address.city.name = "New Y
ork") or (x.name = "ADA")))) (Hotels)


//#### Query 6 ####//

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR
((x.address.number=1910) AND (x.stars>=4))) (Hotels)

Region Top Goal optimize.  Input query is:
```

```
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.stars >= 4)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.stars >= 4)))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) o
r ((x.address.number = 1910) and (x.stars >= 4)))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.stars >= 4)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.stars >= 4)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.number = 1910
) and (x.stars >= 4)))

SR_CNForm customattain
entring rule6 -- (a*b)+(c*d) --> (a+c)*((b+c)*((a+d)*(b+d)))
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.number
= 1910) and (x.stars >= 4)))
Output: (((x.address.city.name = "New York") or (x.address.number = 1910)) and (((x.name
 = "ADA") or (x.address.number = 1910)) and (((x.address.city.name = "New York") or (x.s
tars >= 4)) and ((x.name = "ADA") or (x.stars >= 4)))))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.city.name = "New York") or (x.address.number = 1910))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.address.number = 1910)
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.address.number = 1910))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.name = "ADA") or (x.address.number = 1910))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.number = 1910)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.number = 1910) or (x.name = "ADA"))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.city.name = "New York") or (x.stars >= 4))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.city.name = "New York")
query 1: (x.stars >= 4)
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.city.name = "New York") or (x.stars >= 4))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.name = "ADA") or (x.stars >= 4))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.stars >= 4)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.stars >= 4) or (x.name = "ADA"))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 (((x.address.city.name = "New York") or (x.address.number = 1910)) and (((x.address.numb
er = 1910) or (x.name = "ADA")) and (((x.address.city.name = "New York") or (x.stars >= 4
)) and ((x.stars >= 4) or (x.name = "ADA")))))

SR_simpAND customattain
query 0: ((x.stars >= 4) or (x.name = "ADA"))
query 1: ((x.address.city.name = "New York") or (x.address.number = 1910))
query 2: ((x.address.number = 1910) or (x.name = "ADA"))
query 3: ((x.address.city.name = "New York") or (x.stars >= 4))
Region SimpAND Goal and.  Return queries are:
query 0: (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.name = "New York") o
r (x.address.number = 1910)) and (((x.address.number = 1910) or (x.name = "ADA")) and ((x
.address.city.name = "New York") or (x.stars >= 4)))))

Region CNF Goal CNF.  Return queries are:
query 0: (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.name = "New York") o
r (x.address.number = 1910)) and (((x.address.number = 1910) or (x.name = "ADA")) and ((x
.address.city.name = "New York") or (x.stars >= 4)))))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.n
ame = "New York") or (x.address.number = 1910)) and (((x.address.number = 1910) or (x.nam
e = "ADA")) and ((x.address.city.name = "New York") or (x.stars >= 4)))))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.n
ame = "New York") or (x.address.number = 1910)) and (((x.address.number = 1910) or (x.nam
e = "ADA")) and ((x.address.city.name = "New York") or (x.stars >= 4)))))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.stars >= 4) or (x.name = "ADA")) and (((x.address.city.n
ame = "New York") or (x.address.number = 1910)) and (((x.address.number = 1910) or (x.nam
e = "ADA")) and ((x.address.city.name = "New York") or (x.stars >= 4)))))) (Hotels)


//#### Query 7 ####//

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR
((x.address.number=1910) AND (x.address.city.name="New York"))) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.add
ress.number = 1910) and (x.address.city.name = "New York")))) (Hotels)
```

```
Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.address.city.name = "New York")))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) o
r ((x.address.number = 1910) and (x.address.city.name = "New York")))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.address.city.name = "New York")))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.number = 1910) and (x.address.city.name = "New York")))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.number = 1910
) and (x.address.city.name = "New York")))

SR_CNForm customattain
entring rule7 -- (a*b)+(a*c) --> a*(b+c)
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.number
= 1910) and (x.address.city.name = "New York")))
Output: ((x.address.city.name = "New York") and ((x.name = "ADA") or (x.address.number =
 1910)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.city.name = "New York")

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.name = "ADA") or (x.address.number = 1910))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.number = 1910)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.number = 1910) or (x.name = "ADA"))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name = "ADA"))
)

SR_simpAND customattain
query 0: (x.address.city.name = "New York")
query 1: ((x.address.number = 1910) or (x.name = "ADA"))
Region SimpAND Goal and.  Return queries are:
query 0: ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name =
"ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name =
"ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
1910) or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
1910) or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
1910) or (x.name = "ADA")))) (Hotels)


//#### Query 8 ####//

select(lambda(x)((x.address.number=5) OR ((x.name="ADA") AND
(x.address.number=5)))) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) ((x.address.number = 5) or ((x.name = "ADA") and (x.address.number =
5)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((x.address.number = 5) or ((x.name = "ADA") and (x.address.number =
5)))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((x.address.number = 5) or ((x.name = "ADA") and (x.address.n
umber = 5)))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((x.address.number = 5) or ((x.name = "ADA") and (x.address.number =
5)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((x.address.number = 5) or ((x.name = "ADA") and (x.address.number =
5)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
```

```
 ((x.address.number = 5) or ((x.name = "ADA") and (x.address.number = 5)))

SR_CNForm customattain
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.number = 5)

Region Simplify OR Goal or.  No return queries.

Region CNF Goal CNF.  Return queries are:
query 0: (x.address.number = 5)

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (x.address.number = 5)) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.address.number = 5)) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.address.number = 5)) (Hotels)


//#### Query 9 ####//

select (lambda(x) ((x.stars!=3) AND (x.stars>=3)) AND (x.stars<5)) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))) (Ho
tels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'and'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))

SR_CNForm customattain
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 3) and (x.stars >= 3)) and (x.stars < 5))
Output: ((x.stars < 5) and ((x.stars != 3) and (x.stars >= 3)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.stars < 5)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.stars != 3)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.stars >= 3)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.stars < 5) and ((x.stars != 3) and (x.stars >= 3)))

SR_simpAND customattain
query 0: (x.stars = 4)
Region SimpAND Goal and.  Return queries are:
query 0: (x.stars = 4)

Region CNF Goal CNF.  Return queries are:
query 0: (x.stars = 4)

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (x.stars = 4)) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars = 4)) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars = 4)) (Hotels)


//#### Query 10 ####//

select (lambda(x) ((x.stars!=3) OR (x.stars>=3)) OR (x.stars<5)) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))) (Hotel
s)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
```

```
 (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))

SR_CNForm customattain
entring rule2 -- (a+b)+c --> c+(a+b)
Input: (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 3) or (x.stars >= 3)) or (x.stars < 5))
Output: ((x.stars < 5) or ((x.stars != 3) or (x.stars >= 3)))
Output: ((x.stars < 5) or ((x.stars != 3) or (x.stars >= 3)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars < 5) or ((x.stars != 3) or (x.stars >= 3)))

query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Region CNF Goal CNF.  Return queries are:
query 0: true

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)


//#### Query 11 ####//

select (lambda(x) (x.stars!=3) OR ((x.stars>=3) AND (x.stars<5))) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))) (Hot
els)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))

SR_CNForm customattain
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars != 3) or ((x.stars >= 3) and (x.stars < 5)))
Output: (((x.stars != 3) or (x.stars >= 3)) and ((x.stars != 3) or (x.stars < 5)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars != 3) or (x.stars >= 3))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars != 3) or (x.stars < 5))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Region CNF Goal CNF.  Return queries are:
query 0: true

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)


//#### Query 12 ####//

select (lambda(x) ((x.stars!=3) AND (x.stars>=3)) OR (x.stars<5)) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))) (Hote
ls)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))
```

```
SR_CNForm customattain
entring rule4 -- (c*d)+a --> a+(c*d) --> rule3
Input: (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 3) and (x.stars >= 3)) or (x.stars < 5))
Output: ((x.stars < 5) or ((x.stars != 3) and (x.stars >= 3)))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars < 5) or ((x.stars != 3) and (x.stars >= 3)))
Output: (((x.stars < 5) or (x.stars != 3)) and ((x.stars < 5) or (x.stars >= 3)))
Output: (((x.stars < 5) or (x.stars != 3)) and ((x.stars < 5) or (x.stars >= 3)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars < 5) or (x.stars != 3))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars < 5) or (x.stars >= 3))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Region CNF Goal CNF.  Return queries are:
query 0: true

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) true) (Hotels)


//#### Query 13 ####//

select (lambda(x) ((x.stars!=2) OR (x.stars>=3)) AND (x.stars<2)) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))) (Hot
els)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'and'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))

SR_CNForm customattain
entring rule5 -- (c+d)*a --> a*(c+d)
Input: (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 2) or (x.stars >= 3)) and (x.stars < 2))
Output: ((x.stars < 2) and ((x.stars != 2) or (x.stars >= 3)))
Output: ((x.stars < 2) and ((x.stars != 2) or (x.stars >= 3)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.stars < 2)

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars != 2) or (x.stars >= 3))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.stars != 2)
Region Simplify OR Goal or.  Return queries are:
query 0: (x.stars != 2)

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.stars < 2) and (x.stars != 2))

SR_simpAND customattain
query 0: (x.stars < 2)
Region SimpAND Goal and.  Return queries are:
query 0: (x.stars < 2)

Region CNF Goal CNF.  Return queries are:
query 0: (x.stars < 2)

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (x.stars < 2)) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars < 2)) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars < 2)) (Hotels)


//#### Query 14 ####//

select (lambda(x) ((x.stars!=2) AND (x.stars<3)) OR ((x.stars!=3) AND
(x.stars<4)) OR ((x.stars<=5) AND (x.stars>=2))) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars
< 4))) or ((x.stars <= 5) and (x.stars >= 2)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars
< 4))) or ((x.stars <= 5) and (x.stars >= 2)))) (Hotels)
```

```
Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and
(x.stars < 4))) or ((x.stars <= 5) and (x.stars >= 2)))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars
 < 4))) or ((x.stars <= 5) and (x.stars >= 2)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '!='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<'
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '<='
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '>='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars
 < 4))) or ((x.stars <= 5) and (x.stars >= 2)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 ((((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars < 4))) or ((x.star
s <= 5) and (x.stars >= 2)))

SR_CNForm customattain
entring rule6 -- (a*b)+(c*d) --> (a+c)*((b+c)*((a+d)*(b+d)))
Input: (((x.stars != 2) and (x.stars < 3)) or ((x.stars != 3) and (x.stars < 4)))
Output: (((x.stars != 2) or (x.stars != 3)) and (((x.stars < 3) or (x.stars != 3)) and (
((x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule6 -- (a*b)+(c*d) --> (a+c)*((b+c)*((a+d)*(b+d)))
Input: ((((x.stars != 2) or (x.stars != 3)) and (((x.stars < 3) or (x.stars != 3)) and (
((x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4))))) or ((x.stars <
= 5) and (x.stars >= 2)))
Output: ((((x.stars != 2) or (x.stars != 3)) or (x.stars <= 5)) and (((((x.stars < 3) or
 (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars <
 4)))) or (x.stars <= 5)) and ((((x.stars != 2) or (x.stars != 3)) or (x.stars >= 2)) an
d ((((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((x.sta
rs < 3) or (x.stars < 4)))) or (x.stars >= 2)))))
entring rule2 -- (a+b)+c --> c+(a+b)
Input: (((x.stars != 2) or (x.stars != 3)) or (x.stars <= 5))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 2) or (x.stars != 3)) or (x.stars <= 5))
Output: ((x.stars <= 5) or ((x.stars != 2) or (x.stars != 3)))
Output: ((x.stars <= 5) or ((x.stars != 2) or (x.stars != 3)))
entring rule4 -- (c*d)+a --> a+(c*d) --> rule3
Input: ((((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((
x.stars < 3) or (x.stars < 4)))) or (x.stars <= 5))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: ((((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((
x.stars < 3) or (x.stars < 4)))) or (x.stars <= 5))
Output: ((x.stars <= 5) or (((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x
.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars <= 5) or (((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.
stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
Output: (((x.stars <= 5) or ((x.stars < 3) or (x.stars != 3))) and ((x.stars <= 5) or (((
x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
Output: (((x.stars <= 5) or ((x.stars < 3) or (x.stars != 3))) and ((x.stars <= 5) or (((
x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars <= 5) or (((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.s
tars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
Output: (((x.stars <= 5) or ((x.stars < 3) or (x.stars != 3))) and ((x.stars <= 5) or (((
x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars <= 5) or ((x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.sta
rs < 4))))
Output: (((x.stars <= 5) or ((x.stars != 2) or (x.stars < 4))) and ((x.stars <= 5) or ((x
.stars < 3) or (x.stars < 4))))
entring rule2 -- (a+b)+c --> c+(a+b)
Input: (((x.stars != 2) or (x.stars != 3)) or (x.stars >= 2))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.stars != 2) or (x.stars != 3)) or (x.stars >= 2))
Output: ((x.stars >= 2) or ((x.stars != 2) or (x.stars != 3)))
Output: ((x.stars >= 2) or ((x.stars != 2) or (x.stars != 3)))
entring rule4 -- (c*d)+a --> a+(c*d) --> rule3
Input: ((((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((x
.stars < 3) or (x.stars < 4)))) or (x.stars >= 2))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: ((((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.stars < 4)) and ((x
.stars < 3) or (x.stars < 4)))) or (x.stars >= 2))
Output: ((x.stars >= 2) or (((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.
stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars >= 2) or (((x.stars < 3) or (x.stars != 3)) and (((x.stars != 2) or (x.s
tars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
Output: (((x.stars >= 2) or ((x.stars < 3) or (x.stars != 3))) and ((x.stars >= 2) or (((
x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
Output: (((x.stars >= 2) or ((x.stars < 3) or (x.stars != 3))) and ((x.stars >= 2) or (((
x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.stars < 4)))))
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((x.stars >= 2) or (((x.stars != 2) or (x.stars < 4)) and ((x.stars < 3) or (x.sta
rs < 4))))
Output: (((x.stars >= 2) or ((x.stars != 2) or (x.stars < 4))) and ((x.stars >= 2) or ((x
.stars < 3) or (x.stars < 4))))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars <= 5) or ((x.stars != 2) or (x.stars != 3)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars <= 5) or ((x.stars < 3) or (x.stars != 3)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars <= 5) or ((x.stars != 2) or (x.stars < 4)))
```

```
SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars <= 5) or ((x.stars < 3) or (x.stars < 4)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.stars <= 5)
Region Simplify OR Goal or.  Return queries are:
query 0: (x.stars <= 5)

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars >= 2) or ((x.stars != 2) or (x.stars != 3)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars >= 2) or ((x.stars < 3) or (x.stars != 3)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars >= 2) or ((x.stars != 2) or (x.stars < 4)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars >= 2) or ((x.stars < 3) or (x.stars < 4)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: true
Region Simplify OR Goal or.  Return queries are:
query 0: true

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 (true and (x.stars <= 5))

SR_simpAND customattain
query 0: (x.stars <= 5)
Region SimpAND Goal and.  Return queries are:
query 0: (x.stars <= 5)

Region CNF Goal CNF.  Return queries are:
query 0: (x.stars <= 5)

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (x.stars <= 5)) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars <= 5)) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (x.stars <= 5)) (Hotels)


//#### Query 15 ####//

select(lambda(x)((x.address.city.name="New York") AND (x.name="ADA")) OR
((x.address.city.name="New York") AND (x.address.number=1910)) OR ((x.address.city.name="
New York") AND (x.name="ADA"))) (Hotels)

Region Top Goal optimize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.city.name = "New York") and (x.address.number = 1910))) or ((x.address.city.name =
"New York") and (x.name = "ADA")))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.city.name = "New York") and (x.address.number = 1910))) or ((x.address.city.name =
"New York") and (x.name = "ADA")))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((((x.address.city.name = "New York") and (x.name = "ADA")) o
r ((x.address.city.name = "New York") and (x.address.number = 1910))) or ((x.address.city
.name = "New York") and (x.name = "ADA")))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.city.name = "New York") and (x.address.number = 1910))) or ((x.address.city.name =
"New York") and (x.name = "ADA")))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.ad
dress.city.name = "New York") and (x.address.number = 1910))) or ((x.address.city.name =
"New York") and (x.name = "ADA")))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
```

```
 ((((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.city.name =
"New York") and (x.address.number = 1910))) or ((x.address.city.name = "New York") and (
x.name = "ADA")))

SR_CNForm customattain
entring rule7 -- (a*b)+(a*c) --> a*(b+c)
Input: (((x.address.city.name = "New York") and (x.name = "ADA")) or ((x.address.city.na
me = "New York") and (x.address.number = 1910)))
Output: ((x.address.city.name = "New York") and ((x.name = "ADA") or (x.address.number =
 1910)))
entring rule7 -- (a*b)+(a*c) --> a*(b+c)
Input: (((x.address.city.name = "New York") and ((x.name = "ADA") or (x.address.number =
 1910))) or ((x.address.city.name = "New York") and (x.name = "ADA")))
Output: ((x.address.city.name = "New York") and (((x.name = "ADA") or (x.address.number
= 1910)) or (x.name = "ADA")))
entring rule2 -- (a+b)+c --> c+(a+b)
Input: (((x.name = "ADA") or (x.address.number = 1910)) or (x.name = "ADA"))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: (((x.name = "ADA") or (x.address.number = 1910)) or (x.name = "ADA"))
Output: ((x.name = "ADA") or ((x.name = "ADA") or (x.address.number = 1910)))
Output: ((x.name = "ADA") or ((x.name = "ADA") or (x.address.number = 1910)))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 (x.address.city.name = "New York")

Region Simplify OR Goal or.  No return queries.

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.name = "ADA") or ((x.name = "ADA") or (x.address.number = 1910)))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.number = 1910)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.number = 1910) or (x.name = "ADA"))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name = "ADA"))
)

SR_simpAND customattain
query 0: (x.address.city.name = "New York")
query 1: ((x.address.number = 1910) or (x.name = "ADA"))
Region SimpAND Goal and.  Return queries are:
query 0: ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name =
 "ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: ((x.address.city.name = "New York") and ((x.address.number = 1910) or (x.name =
 "ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
 1910) or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
 1910) or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) ((x.address.city.name = "New York") and ((x.address.number =
 1910) or (x.name = "ADA")))) (Hotels)

//#### Query 16 ####//
select(lambda(x)((((x.address.city.name="New York") AND false) OR (x.name="ADA") OR ((x.s
tars=4) AND (x.address.number=1910))))) (Hotels)
Region Top Goal optimize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and false) or (x.name = "ADA"))
 or ((x.stars = 4) and (x.address.number = 1910)))) (Hotels)

Branching for goal 'normalize': First, 1.
Region Normalize Goal normalize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and false) or (x.name = "ADA"))
 or ((x.stars = 4) and (x.address.number = 1910)))) (Hotels)

Region Normalize Goal normalize.  Return queries are:
query 0: select (lambda (x) ((((x.address.city.name = "New York") and false) or (x.name =
 "ADA")) or ((x.stars = 4) and (x.address.number = 1910)))) (Hotels)

Branching for goal 'optimize': First, 1.
Region Optimizer Goal optimize.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and false) or (x.name = "ADA"))
 or ((x.stars = 4) and (x.address.number = 1910)))) (Hotels)

WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'city'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'name'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'or'
WARNING: Region 'Optimizer' does not know about goal 'stars'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'address'
WARNING: Region 'Optimizer' does not know about goal 'number'
WARNING: Region 'Optimizer' does not know about goal '='
WARNING: Region 'Optimizer' does not know about goal 'and'
WARNING: Region 'Optimizer' does not know about goal 'or'
Branching for goal 'select': First, 3.
Region Select Goal select.  Input query is:
 select (lambda (x) ((((x.address.city.name = "New York") and false) or (x.name = "ADA"))
 or ((x.stars = 4) and (x.address.number = 1910)))) (Hotels)

SR_Select:  did transforms
Branching for goal 'CNF': First, 3.
Region CNF Goal CNF.  Input query is:
 ((((x.address.city.name = "New York") and false) or (x.name = "ADA")) or ((x.stars = 4)
and (x.address.number = 1910)))

SR_CNForm customattain
entring rule3 -- a+(c*d) --> (a+c)*(a+d)
Input: ((false or (x.name = "ADA")) or ((x.stars = 4) and (x.address.number = 1910)))
Output: (((false or (x.name = "ADA")) or (x.stars = 4)) and ((false or (x.name = "ADA"))
or (x.address.number = 1910)))
entring rule2 -- (a+b)+c --> c+(a+b)
Input: ((false or (x.name = "ADA")) or (x.stars = 4))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: ((false or (x.name = "ADA")) or (x.stars = 4))
Output: ((x.stars = 4) or (false or (x.name = "ADA")))
Output: ((x.stars = 4) or (false or (x.name = "ADA")))
entring rule2 -- (a+b)+c --> c+(a+b)
Input: ((false or (x.name = "ADA")) or (x.address.number = 1910))
entring rule1 -- (a*b)*c --> c*(a*b)
Input: ((false or (x.name = "ADA")) or (x.address.number = 1910))
Output: ((x.address.number = 1910) or (false or (x.name = "ADA")))
Output: ((x.address.number = 1910) or (false or (x.name = "ADA")))
Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.stars = 4) or (false or (x.name = "ADA")))
```

```
SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.stars = 4)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.stars = 4) or (x.name = "ADA"))

Branching for goal 'or': First, 3.
Region Simplify OR Goal or.  Input query is:
 ((x.address.number = 1910) or (false or (x.name = "ADA")))

SR_SimpAND::CustomAttain.  Delaying until variable defined.
query 0: (x.address.number = 1910)
query 1: (x.name = "ADA")
Region Simplify OR Goal or.  Return queries are:
query 0: ((x.address.number = 1910) or (x.name = "ADA"))

Branching for goal 'and': First, 3.
Region SimpAND Goal and.  Input query is:
 (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number = 1910) or (x.name = "ADA")
))

SR_simpAND customattain
query 0: ((x.stars = 4) or (x.name = "ADA"))
query 1: ((x.address.number = 1910) or (x.name = "ADA"))
Region SimpAND Goal and.  Return queries are:
query 0: (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number = 1910) or (x.name
= "ADA")))

Region CNF Goal CNF.  Return queries are:
query 0: (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number = 1910) or (x.name
= "ADA")))

Region Select Goal select.  Return queries are:
query 0: select (lambda (x) (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number
= 1910) or (x.name = "ADA")))) (Hotels)

Region Optimizer Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number
= 1910) or (x.name = "ADA")))) (Hotels)

Region Top Goal optimize.  Return queries are:
query 0: select (lambda (x) (((x.stars = 4) or (x.name = "ADA")) and ((x.address.number
= 1910) or (x.name = "ADA")))) (Hotels)

Exiting Epoq properly. Bye..
there were a peak of 415 OVoid-derived objects created

II. Simplify AND region

org x=4 and x=4
x=4
org x=4 and x=5
false
org x=4 and x!=4
false
org x=4 and x!=5
x=4
org x=4 and x>=3
x=4
org x=4 and x>=5
false
org x=4 and x<=5
x=4
org x=4 and x<=3
false
org x=4 and x>3
x=4
org x=4 and x>4
false
org x=4 and x<5
x=4
org x=4 and x<4
false
org x!=4 and x=4
false
org x!=4 and x=3
x=3
org x!=4 and x!=4
x!=4
org x!=4 and x!=3
x!=4 and x!=3
org x!=4 and x>=4
x > 4
org x!=4 and x>=3
x!=4 and x>=3
org x!=4 and x>=5
x>=5
org x!=4 and x<=4
x < 4
org x!=4 and x<=3
x<=3
org x!=4 and x<=5
x!=4 and x<=5
org x!=4 and x>3
x > 4
org x!=4.0 and x>3.0
x!=4.0 and x>3.0
org x!=4 and x>5
x>5
org x!=4 and x<3
x<3
org x!=4 and x<5
x < 4
org x!=4.0 and x<5.0
x!=4.0 and x<5.0
org x>=4 and x=3
false
org x>=4 and x=5
x=5
org x>=4 and x!=4
x > 4
org x>=4 and x!=3
x>=4
org x>=4 and x!=5
x>=4 and x!=5
org x>=4 and x>=3
x>=4
org x>=4 and x<=4
x = 4
org x>=4 and x<=3
false
org x>=4 and x<=5
x>=4 and x<=5
org x>=4 and x>3
x>=4
org x>=4 and x>5
x>5
org x>=4 and x<3
false
org x>=4 and x<5
x = 4
org x>=4.0 and x<5.0
x>=4.0 and x<5.0
org x>=4 and x<6
```

```
x>=4 and x<6
org x<=4 and x=3
x=3
org x<=4 and x=5
false
org x<=4 and x!=4
x < 4
org x<=4 and x!=3
x<=4 and x!=3
org x<=4 and x!=5
x<=4
org x<=4 and x>=4
x = 4
org x<=4 and x>=3
x<=4 and x>=3
org x<=4 and x>=5
false
org x<=4 and x<=3
x<=3
org x<=4 and x<=5
x<=4
org x<=4 and x>5
false
org x<=4 and x>3
x = 4
org x<=4.0 and x>3.0
x<=4.0 and x>3.0
org x<=4 and x>2
x<=4 and x>2
org x<=4 and x<3
x<3
org x<=4 and x<5
x<=4
org x>4 and x=5
x=5
org x>4 and x=3
false
org x>4 and x!=3
x>4
org x>4 and x!=5
x > 5
org x>4.0 and x!=5.0
x>4.0 and x!=5.0
org x>4 and x>=5
x>=5
org x>4 and x>=3
x>4
org x>4 and x<=3
false
org x>4 and x<=5
x = 5
org x>4.0 and x<=5.0
x>4.0 and x<=5.0
org x>4 and x<=6
x>4 and x<=6
org x>4 and x>3
x>4
org x>4 and x>5
x>5
org x>4 and x<3
false
org x>4 and x<5
false
org x>4.0 and x<5.0
x>4.0 and x<5.0
org x>4 and x<6
x = 5
org x>4.0 and x<6.0
x>4.0 and x<6.0
org x>4 and x<7
x>4 and x<7
org x<4 and x=3
x=3
org x<4 and x=5
false
org x<4 and x!=3
x < 3
org x<4.0 and x!=3.0
x<4.0 and x!=3.0
org x<4 and x!=5
x<4
org x<4 and x>=5
false
org x<4 and x>=3
x = 3
org x<4.0 and x>=3.0
x<4.0 and x>=3.0
org x<4 and x>=2
x<4 and x>=2
org x<4 and x>5
false
org x<4 and x>3
false
org x<4.0 and x>3.0
x<4.0 and x>3.0
org x<4 and x>2
x = 3
org x<4.0 and x>2.0
x<4.0 and x>2.0
org x<4 and x>1
x<4 and x>1
org x<4 and x<5
x<4
org x<4 and x<3
x<3

III. Simplify OR region

org x=4 or x=4
x=4
org x=4 or x=3
x=4 or x=3
org x=4 or x!=4
true
org x=4 or x!=3
x!=3
org x=4 or x>=3
x>=3
org x=4 or x>=5
x >= 4
org x=4.0 or x>=5.0
x=4.0 or x>=5.0
org x=4 or x>=6
x=4 or x>=6
org x=4 or x<=5
x<=5
org x=4 or x<=3
x <= 4
org x=4.0 or x<=3.0
x=4.0 or x<=3.0
org x=4 or x<=2
x=4 or x<=2
org x=4 or x>4
x >= 4
```

```
org x=4 or x>3
x>3
org x=4 or x>5
x=4 or x>5
org x=4 or x<4
x <= 4
org x=4 or x<5
x<5
org x=4 or x<3
x=4 or x<3
org x!=4 or x=4
true
org x!=4 or x=3
x!=4
org x!=4 or x!=4
x!=4
org x!=4 or x!=3
x!=4 or x!=3
org x!=4 or x>=3
true
org x!=4 or x>=5
x!=4
org x!=4 or x<=5
true
org x!=4 or x<=3
x!=4
org x!=4 or x>5
x!=4
org x!=4 or x>3
true
org x!=4 or x<3
x!=4
org x!=4 or x<5
true
org x>=4 or x=5
x>=4
org x>=4 or x=3
x >= 3
org x>=4.0 or x=3.0
x>=4.0 or x=3.0
org x>=4 or x=2
x>=4 or x=2
org x>=4 or x!=5
true
org x>=4 or x!=3
x!=3
org x>=4 or x>=5
x>=4
org x>=4 or x>=3
x>=3
org x>=4 or x<=5
true
org x>=4 or x<=3
true
org x>=4.0 or x<=3.0
x>=4.0 or x<=3.0
org x>=4 or x<=2
x>=4 or x<=2
org x>=4 or x>3
x>3
org x>=4 or x>5
x>=4
org x>=4 or x<5
true
org x>=4.0 or x<5.0
true
org x>=4 or x<3
x != 3
org x>=4.0 or x<3.0
x>=4.0 or x<3.0
org x>=4 or x<2
x>=4 or x<2
org x<=4 or x=3
x<=4
org x<=4 or x=5
x <= 5
org x<=4.0 or x=5.0
x<=4.0 or x=5.0
org x<=4 or x=6
x<=4 or x=6
org x<=4 or x!=3
true
org x<=4 or x!=5
x!=5
org x<=4 or x>=3
true
org x<=4 or x>=5
true
org x<=4.0 or x>=5.0
x<=4.0 or x>=5.0
org x<=4 or x>=6
x<=4 or x>=6
org x<=4 or x<=3
x<=4
org x<=4 or x<=5
x<=5
org x<=4 or x>3
true
org x<=4 or x>5
x != 5
org x<=4.0 or x>5.0
x<=4.0 or x>5.0
org x<=4 or x>6
x<=4 or x>6
org x<=4 or x<3
x<=4
org x<=4 or x<5
x<5
org x>4 or x=4
x >= 4
org x>4 or x=5
x>4
org x>4 or x=3
x>4 or x=3
org x>4 or x!=3
x!=3
org x>4 or x!=5
true
org x>4 or x>=5
x>4
org x>4 or x>=3
x>=3
org x>4 or x<=5
true
org x>4 or x<=3
x != 4
org x>4.0 or x<=3.0
x>4.0 or x<=3.0
org x>4 or x<=2
x>4 or x<=2
org x>4 or x>5
x>4
org x>4 or x>3
x>3
```

```
org x>4 or x<4
x != 4
org x>4 or x<5
true
org x>4 or x<3
x>4 or x<3
org x<4 or x=4
x <= 4
org x<4 or x=3
x<4
org x<4 or x=5
x<4 or x=5
org x<4 or x!=5
x!=5
org x<4 or x!=3
true
org x<4 or x>=3
true
org x<4 or x>=5
x != 4
org x<4.0 or x>=5.0
x<4.0 or x>=5.0
org x<4 or x>=6
x<4 or x>=6
org x<4 or x<=5
x<=5
org x<4 or x<=3
x<4
org x<4 or x>4
x != 4
org x<4 or x>3
true
org x<4 or x>5
x<4 or x>5
org x<4 or x<3
x<4
org x<4 or x<5
x<5
```