Some ideas on relations between queries

Siegfried Nijssen

May 3, 2001

1 An input/output layout for queries

The following could be a layout for a query:

$$Q_-(\{X_1, \ldots, X_n\}), Q_+(\{X_1, \ldots, X_n\}), \tag{1}$$

where:

- Q_- contains atoms which introduce variables $X_1, \ldots, X_n$, but not reuse them;
- Q_+ contains atoms which only use variables in $X_1, \ldots, X_n$, but not introduce any new variables.

We will investigate whether queries of this layout may constitute a class of queries for which a more efficient refinement operator can be constructed. Before doing so, we will consider the relation of such queries to full-fledged queries. We will keep the following observations in mind:
Question 1: can atoms in any query be reordered to obey to this shape?
Answer: No. This example illustrates this:

$$p(X, Y), q(Y, Z), r(X, Z). \tag{2}$$

Whatever atom is used first, one variable remains to be introduced which can not be introduced without using an already existing variable.
Question 2: can every query be adapted such that it can be rewritten in this shape?
Answer: in a sense, yes, by introducing one predicate which introduces variables and is true for any variable value:

$$t(X), t(Y), t(Z), p(X, Y), q(Y, Z), r(X, Z). \tag{3}$$

2 Reordering is NP-complete

We will derive this result:

> **To determine whether atoms in a query can be reordered to obtain an input/output layout, is NP-complete to solve**.

Furthermore, we will show that an i/o reordering is not unique. If for a given set of isomorph queries one want to determine one unique representant, these observations lead to the conclusion that the following mechanism is not sensible:

1. given a query Q, determine whether its atoms can be reordered;
2. if so, apply the possibly efficient algorithm to determine whether it is in some normal form;
3. otherwise, apply an inefficient algorithm.

The main reason for looking into this mechanism was to gain more insight in the properties of i/o queries.

In the proof, we will use this theorem:

> **One-in-three 3SAT is NP complete**
> **Instance: set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.**
> **Question: is there a truth assignment for U such that each clause in C has *exactly one* true literal?**
> **This problem is NP-complete [Schaeffer, 1978b].**

We will reduce this 1-in-3SAT problem to our problem, hereby showing that our i/o reordering problem can not be solved efficiently either. As intermediate step, we will use the following problem:

> **One-COVER**
> **Instance: a set of integers I and a collection of subsets $C \subseteq 2^I$.**
> **Question: is there a set $T \subseteq I$, such that for all $c \in C : |c \cap T| = 1$?**

Example Given $I = \{1, 2, 3, 4, 5, 6\}, C = \{\{1, 2, 3, 4\}, \{1, 3, 5\}, \{2, 4, 6\}, \{2, 5, 6\}\}$, such a subset exists: $T = \{4, 5\}$. Another solution is $T = \{3, 6\}$. □

This problem can be reduced to our problem by the following construction:

1. assign a variable to each set in C;
2. let each element $i \in I$ correspond to a predicate, which has as parameters the sets in C that contain i.

Example The previous example reduces to this query: $p_1(X_1, X_2), p_2(X_1, X_3, X_4), p_3(X_1, X_2), p_4(X_1, X_3), p_5(X_2, X_4), p_6(X_3, X_4)$. A set of outputting atoms is $p_4(X_1, X_3), p_5(X_2, X_4)$, which corresponds to the set $\{4, 5\}$; also $p_3(X_1, X_2), p_6(X_3, X_4)$ would be a solution. □

Generally, the correctness of this reduction can be seen as follows:

1. two outputting atoms can not introduce the same variable; thus, the two corresponding integers can not occur in the same set: for given sets c and T, $|c \cap T| \leq 1$;

2. as a variable must be introduced by an outputting atom, the corresponding set c must contain at least one element i in T ($|c \cap T| \geq 1$).

It can easily be seen that the mapping can be computed in polynomial time. The atom ordering problem is thus NP-complete if One-COVER is NP-complete. We will show that One-COVER is NP-complete by defining a polynomial mapping from 1-in-3SAT to One-COVER. Given a set of clauses C containing variables $X_1, \ldots, X_n$, construct the following sets:

1. $\{X_i, \neg X_i\}$ for every $1 \leq i \leq n$;
2. $\{L_{c1}, L_{c2}, L_{c3}\}$ for every $c \in C$, where L_{cj} is the j-th literal of c.

The correctness of this mapping is easily shown:

1. the first n sets take care that every variable gets exactly one value;
2. the other sets take care that in every clause, there is exactly one literal that is true.

This shows that in the general case it is not efficiently possible to decide whether atoms can be reordered: as the correctness of a solution can be checked polynomially, the problem is NP-complete.

Of course, for some special cases, a reordering can still easily be found. We will mention one special example here:

Example Consider a query in which every variable occurs at most twice. This problem can be reduced to a one-COVER problem in which every set has size one or two. By interpreting sets of size two as edges in a graph, this problem reduces to determining whether a graph is bipartite (with all singleton sets in one partition), which is easily checked. □

This gives rise to the impression that the difficulty of the problem increases with the number of times one variable occurs.

3 Ranking i/o queries

We will use the partition of atoms in input and output atoms to order queries. To clearify our discussion, we first review the standard ordering of atoms in PROLOG. The criteria used to order atoms in PROLOG are:

- first, the arity of the atoms (from small to large);
- secondly, the alphabetical order of the predicate names;
- thirdly, the alphabetical order of the constant names in the atoms;
- fourhtly, the age of variables in the atoms (from old to young).

Example The following list of atoms is in order:

$$p(X), q(Y), r(X, a), r(X, b), s(X, Y), s(Y, X), \tag{4}$$

as (among others) p and q have lower arity than r and s; p is lower than q; a is lower than b; X is introduced earlier than Y. □

Not every list of atoms can be ordered uniquely.

Example This list of atoms can not be ordered using the ordering rules:

$$p(X), p(Y), s(X, Y), \tag{5}$$

as both variables X and Y are new and thus have no age. □

If every variable has an age, the order of the atoms is uniquely defined. In the i/o layout, this means that all input atoms can be ordered uniquely given an order of the output atoms, as this order defines the age of all variables.

The order of atoms can also be used to rank queries.

Example The following holds:

$$t(X), t(Y), p(X, Y), q(X, Y) < t(X), t(Y), p(Y, X), p(X, Y), \tag{6}$$

as the first unequal atoms from the left have this relation: $p(X, Y) < p(Y, X)$. □

For any set of input and output atoms, this can be used to define a normal form:

> **Given a query $Q = Q_-(\{X_1, \ldots, X_n\}), Q_+(\{X_1, \ldots, X_n\})$, a normal order of atoms is:**
>
> 1. **for Q_+: the unique order resulting from the choice of Q_-;**
> 2. **for Q_-: the order which results in the lowest atom order for Q_+.**

Example The following query is not in normal form (age is given in superscripts):

$$t(X^1), t(Y^2), t(Z^3), p(X^1, X^1), p(Y^2, Z^3), p(Z^3, X^1), \tag{7}$$

but the following reordering is:

$$t(X^1), t(Z^2), t(Y^3), p(X^1, X^1), p(Z^2, X^1), p(Y^3, Z^2). \tag{8}$$

□

Example Consider these two queries:

$$t(X), t(Y), t(Z), p(X, X), p(Y, X), p(Y, Z), p(Z, X), p(Z, Y) \tag{9}$$

$$t(X), t(Z), t(Y), p(X, X), p(Z, X), p(Z, Y), p(Y, X), p(Y, Z) \tag{10}$$

One query is a reordering of the other and both are in normal form. However, also the substitution $\{Y/Z, Z/Y\}$ yields one query from the other. It appears that there is more

than one normal form iff there is a substitution which maps the query to itself (which is a renaming of variables). If variables are always given a name according to their age, this problem disappears and the normal form is unique. Both queries become:

$$t(X_1), t(X_2), t(X_3), t(X_1, X_1), p(X_2, X_1), p(X_2, X_3), p(X_3, X_1), p(X_3, X_2) \quad (11)$$

□

We will now show the correctness of the following statement:

> **To determine the normal order of a query, is at least isomorphism-complex.**

This can be seen using the following observations:

1. from any query an i/o query can be constructed by prefixing it with atoms of a special predicate that introduces variables (see question 2);
2. if we could reduce two i/o queries efficiently to a normal order, we could use this reduction algorithm to determine whether the two queries are isomorphisms;
3. isomorphism of queries is isomorphism-complex to compute; currently no algorithm in P is known which performs this task.

However, this does *not* immediately mean that it is isomorphism-complex to determine whether an atom order is in normal form. A polynomial algorithm is searched for which for any query not in normal form discovers some query which is lower. The following example illustrates that this is not straightforward.

Example The normal form can be obtained from any query by sequencing permutations of size 2. If there is always a chain of such permutations which decreases the rank monotonically, then the problem would reduce to finding a first element of the chain. However, this query demonstrates that this approach is too greedy:

$$t(X_1), t(X_2), t(X_3), p(X_1, X_1), p(X_1, X_2), p(X_2, X_2), p(X_3, X_1), p(X_3, X_3). \quad (12)$$

It can be shown that every swap of 2 t atoms leads to a higherly ranked query, although the query is not the normal form:

$$t(X_3), t(X_1), t(X_2), p(X_3, X_3), p(X_3, X_1), p(X_1, X_1), p(X_1, X_2), p(X_2, X_2). \quad (13)$$

□

4 Further Items To Be Considered

- find a polynomial algorithm or show that the problem is isomorphism-complex;
- define the link to Farmer tries and the "normal form" used there; advantages/disadvantages;
- look into object orientation;
- meaning of the theorem proofer.

References

[Schaeffer, 1978b] Schaeffer. *The complexity of Satisfiability problems*, Proceedings of the 10th Annual ACM Symposium on Theory of Computing, Association for Computing Machines, New York, 1978, 216-226 (A1.1;A9.1;A9.2)
[Garey, 1979] Garey, R., David S. Johnson. *Computers and intractability: A guide to the Theory of NP-completeness.* W.H. Freeman and company, San Francisco, 1979.