1 Introduction

A deductive database system consists of a set of explicitly defined relations (facts in logic programming terms) and a set of Horn clause rules which define the implicit relations (rules). A query in such a system consists of a conjunction of atoms, just as in Prolog. However a major difference between Prolog and most deductive database systems is that in the latter, bottom-up execution is generally used, rather than a top-down method such as SLD-resolution. As a consequence, the implementation techniques and optimization methods for deductive databases are somewhat different from those used in Prolog.

An important feature of top-down methods is that the current goal is used to generate new goals until no more can be found (i.e. until the new goals generated are facts). In this approach. the unification mechanism ensures that only goals relevant to the original one are generated, as clauses whose head does not match the current sub-goal are ignored, and the solution space is search one tuple at a time, so that if all solutions are desired, backtracking is used to find each one of them on demand. On the other hand, a bottom-up method starts with the facts and continually applies the rules until no more new facts are generated. In this way all solutions are found in the normal computations, and so no backtracking is used. A naive implementation of this approach will generate all facts derivable from the program, which is clearly wasteful. One way to avoid such waste is to use optimization methods such as *magic sets*. Such methods usually involve re-writing the rules of the database to take advantage of some extra information, such as the adornments of the definition of the relation. This information is used to ensure that only "relevant" work is done. For example, given the query

?- $t(1, Y)$

where t is the usual trasitive closure of the base relation p, it is clearly wasteful to calculate the entire relation t, and then choose only those tuples whose first attribute is 1. Instead, we wish to restrict the computation to only those tuples which are relevant to this query. The magic set transformation is one way of ensuring that only relevant tuples are computed, and the transformed version of the program is given below.

$t(X, Y)$:- $m(X), p(X, Y)$.
$t(X, Y)$:- $m(X), p(X, Z), t(Z, Y)$.
$m(Z)$:- $m(X), p(X, Z)$.
$m(1)$.

Note that we may think of this transformation as passing constraints from the head to the body, in that the constraints implied by restricting attention to the part of the a relation whose first attribute is 1 are calculated and then passed into the p relation (assuming a left-to-right evaluation). In fact it can be shown that the second rule above is redundant, and hence the computation consists only of calculating the implied constraints (i.e. finding the m relation), and joining this relation with p. In this way we only generate answers which are relevant to the query asked.

The important point to note about this example is that we may consider the magic set transformation as a particular means of *constraint propagation*, i.e. a method of determining a given constraint (in this case that the output should be restricted to tuples relevant to the query), and applying the constraint so inferred in such a way that it its effect is maximised. In the case of magic sets, the information inferred from the query is that certain variables are bound to particular values, and so the constraints propagated may be described as "binding" constraints. Given our knowledge of the computation mechanism, we know that one such constraint will generate more binding constraints in a known manner, i.e. in a way specified by the program. It seems reasonable to assume that other constraints may be propagated in a similar manner. For example, a constraint such as $X > 1$ is not a binding constraint, but rather a "filtering" constraint, in that certain values of X make this constraint true and others do not. It was shown in [4] how such constraints may be propagated. Another possibility is to use integrity constraints to restrict the search space of the query [2].

Another kind of constraint that may be propagated are those which arise from multiple occurrences of the same variable. For example, if t_1 and t_2 are two transitive closure relations, consider the query

$$? - t_1(X, Y), t_2(Y, Z)$$

If the second attribute of t_1 and the first attribute of t_2 both have large domains, but for which the intersection of the two is small, then a naive left-to-right evaluation will generate many values of Y which are irrelevant to the query, as they fall outside the domain of the first attribute of t_2. In this case we wish to use the domain information to constrain the generation of tuples so that, just as in the case for binding constraints, only tuples relevant to the query are generated.

In this paper we show how such constraints may be propagated by the use of *fold* and *unfold* transformations [11, 1]. We also show how some binding patterns not captured by magic sets, such as those discussed in [3], may be captured very naturally by the use of fold/unfold transformations. Hence we show how the use of these simple techniques may be used as a powerful means of constraint propagation, an observation mentioned in [10].

This paper is organised as follows. In Section 2 we give the necessary technical preliminaries, and in the following section we show how multiple variable constraints may be propagated by fold/unfold transformations. In Section 4 we show how some examples of constraints which cannot be captured by magic sets may be similarly reduced, and finally in Section 5 we discuss our conclusions and possibilities for further work.

2 Preliminaries

In this paper a deductive database consists of a finite set of Datalog clauses, i.e. Horn clauses without function symbols.

We consider such a set of clauses as defining a set of relations. *Base relations* are those which are those for which the body of all the clauses defining the relation are empty. *Derived relations* are those for which the body of all the clauses defining the relation are non-empty. Note that without loss of generality, we may assume that all relations are either base relations or derived relations. Furthermore, we assume that all clauses are *range-restricted*, i.e. that all variables which occur in the head of a clause must occur in the body of the clause (note that this means that the base relations must contain only ground terms).

We also use the usual notion of adornment [12] in order to express mode information.

We shall use the default left-to-right *sideways information-passing strategy* (sip) throughout this paper.

The definitions of the fold/unfold transformations are given in [11, 1], and are somewhat involved; for our purposes it will suffice to note that folding replaces a given conjunction of goals in the body of a clause with a call to an equivalent predicate, whereas unfolding "unwinds" the definition of a given predicate by replacing occurrences of the predicate by the body of its definition.

3 Multiple Variable Constraints

Often a constraint may be inferred from multiple occurrences of the same variable. For example, consider the two transitive closure relations below.

$t_1(X, Y)$:- $a(X, Y)$.
$t_1(X, Y)$:- $b(X, Z), t_1(Z, Y)$.

$t_2(X, Y)$:- $c(X, Y)$.
$t_2(X, Y)$:- $d(X, Z), t_2(Z, Y)$.

Given the query

?- $t_1(X, Y), t_2(Y, Z)$

we can obviously proceed in a left-to-right manner by first evaluating all the answers to $t_1(X, Y)$, and then passing these Y values to $t_2(Y, Z)$. However, this is wasteful when the domains of the second attribute of t_1 and the first attribute of t_2 are large, but have a small intersection. In a top-down methodology, we may use intelligent backtracking techniques to minimise the amount of work done. In the bottom-up case, we again use constraint propagation. In particular, we wish to execute the query below

?- $t_1(X, Y), dom1\_t2(Y), t_2(Y, Z)$

where $dom1\_t2(Y)$ is defined as follows.

$dom1\_t2(X)$:- $c(X, Y)$.
$dom1\_t2(Z)$:- $dom1\_t2(X), d(X, Z)$.

Clearly if we can propagate the constraint $dom1\_t2(Y)$ into the definition of $t_1(X, Y)$, then we will only generate answers which are relevant to the second part of the query. We again use fold and unfold transformations to do this. First, we fold the conjunction $t_1(X, Y), dom1\_t2(Y)$ into one relation, so that we ask the query below

?- $t(X, Y), t_2(Y, Z)$

where we have the following clause

$t(X,Y)$:- $t_1(X,Y), dom1\_t2(Y)$.

We then unfold this definition (using that of t_1) to obtain

$t(X,Y)$:- $a(X,Y), dom1\_t2(Y)$.
$t(X,Y)$:- $b(X,Z), t_1(Z,Y), dom1\_t2(Y)$.

which we may rewrite as

$t(X,Y)$:- $a(X,Y), dom1\_t2(Y)$.
$t(X,Y)$:- $b(X,Z), t(Z,Y)$.

Note that this is a generic procedure, in that we can use the same method for *any* filtering constraint for Y: we have not (yet) used any particular properties of $dom1\_t2(Y)$. Clearly we may unfold again, using the definition of $dom1\_t2(Y)$ to obtain the clauses below.

$t(X,Y)$:- $a(X,Y), c(Y,Z)$.
$t(X,Y)$:- $a(X,Y), dom1\_t2(Z), d(Z,Y)$.
$t(X,Y)$:- $b(X,Z), t(Z,Y)$.

At this stage we may use the magic sets transformation to further transform the program. Another possibility is to use an overapproximation of the relevant domain, rather than calculate it exactly. This may in fact be more efficient than the exact calculation of the domain, which may dominate the cost of the computation. The overapproximation case corresponds to using the following definition of $dom1\_t2$:

$dom1\_t2(X)$:- $c(X,Y)$.
$dom1\_t2(Z)$:- $d(X,Z)$.

Clearly this would result in the following definition of t:

$t(X,Y)$:- $a(X,Y), c(Y,Z)$.
$t(X,Y)$:- $a(X,Y), d(Z,Y)$.
$t(X,Y)$:- $b(X,Z), t(Z,Y)$.

In this way the propagation of the constraint only involves base predicates, and hence will presumably be more efficient than the earlier one.

When applying such a propagation in practice, it will be useful to have some heuristics, presumably provided by abstract interpretation techniques, to determine when to use this propagation method, and whether to use the overapproximation or not. Thus empirical evidence will be useful guide for determining when the number of answers is “sufficiently large” and the intersection “sufficiently small” to justify this transformation.

4 "Non-Magic" Constraints

In [3] it is shown how the propagation of some binding constraints, which are not captured by magic sets, may be propagated by using unfolding. In this section we show how two examples of this sort may be performed more simply by the use of folding as well.

The first example is for the relation p below in the mode bbf.

$p(X_1, X, Y)$:- $e(X_1, X, Y)$.
$p(X_1, X, Y)$:- $c(X, Y), p(X_2, X_1, Y_1), d(X_2, Y_1)$.

Note that in the recursive call to p, the first and third arguments are variables local to the body, i.e. do not appear in the head. Hence, the only possible modes for the recursive call are fbf and fff. It is this that stops the propagation of bindings from the head, as even if the relation p is queried in mode bbb, the recursive call will still be fbf, and a second recursive call (if any) will be in mode fff, thus making impossible the propagation of the bindings here by magic sets.

Note also that whilst the variables X_2 and Y_1 are local to the body, there is a "real" constraint here, in that we must have the first and last arguments of both p and d being the same. Thus the constraint is not a binding constraint, but a *filtering* one, in that certain values pass the filter and others don't.

In [3] it is shown how this constraint may be propagated by unfolding, to obtain the clauses

$p(X_1, X, Y)$:- $e(X_1, X, Y)$.
$p(X_1, X, Y)$:- $c(X, Y), e(X_2, X_1, Y_1),$
$\quad d(X_2, Y_1)$.
$p(X_1, X, Y)$:- $c(X, Y), t(X_1, X_2),$
$\quad e(X_3, X_2, Y_1), d(X_3, Y_1)$.
$t(X_1, X_2)$:- $cd(X_1, X_2)$.
$t(X_1, X_2)$:- $cd(X_1, W), t(W, X_2)$.
$cd(X_1, X_2)$:- $c(X_1, Y_1), d(X_2, Y_1)$.

Note that by grouping the calls to c and d into the one relation cd corresponds to moving the original call to d to the left of the recursive call. This is the key to the optimisation, in that more bindings may be found before the recursive call is evaluated.

Another way that this constraint may be propagated is to fold the recursive call to p and the call to d into one relation, say t. This results in the clauses below.

$p(X_1, X, Y)$:- $e(X_1, X, Y)$.
$p(X_1, X, Y)$:- $c(X, Y), t(X_1)$.
$t(X_1)$:- $p(X_2, X_1, Y_1), d(X_2, Y_1)$.

Note that t need only be a unary relation, as X_1 is the only non-local

variable to appear in $p(X_2, X_1, Y_1), d(X_2, Y_1)$. Next we unfold the call to p in the body of t to obtain the clauses below.

$p(X_1, X, Y)$:- $e(X_1, X, Y)$.
$p(X_1, X, Y)$:- $c(X, Y), t(X_1)$.
$t(X_1)$:- $e(X_2, X_1, Y_1), d(X_2, Y_1)$.
$t(X_1)$:- $c(X_1, Y_1), t(X_2), d(X_2, Y_1)$.

Now from the second clause it is easy to see that the binding propagations for c and t are independent, and hence we may easily execute the body of this rule in parallel. Also, note that the last rule may be re-ordered so that the binding for X_1 may be propagated so that if the initial call to t is in mode b, then so is the recursive call to t. This means that we may further optimise this code by the usual magic set transformation in this case. However, a call to t in mode b requires no output, the computation reduces to a test, in that it is only necessary to check that the given argument is in the relation. This is known as an *existential* query, and certain techniques for optimisation of such queries are known [7]. The important point to note is that not only does the use of folding allow us to propagate the constraint more neatly than [3], but also it allows us to easily identify further optimisations, such as the existential one.

Another example given in [3] is the one below.

$p(X, Y, Z)$:- $e(X, Y, Z)$.
$p(X, Y, Z)$:- $c(X_1, X, Y), p(X_1, Y_1, Z_1)$,
$d(Y_1, Z, Z_1)$.

In this case, the constraint is propagated to the clauses below in [3]:

$p(X, Y, Z)$:- $e(X, Y, Z)$.
$p(X, Y, Z)$:- $c(X_1, X, Y), e(X_1, Y_1, Z_1)$,
$d(Y_1, Z, Z_1)$.
$p(X_1, X, Y)$:- $ccdd(X, Y, Z, X_2, Y_2, Z_2)$,
$p(X_2, Y_2, Z_2)$.
$ccdd(X, Y, Z, X_2, Y_2, Z_2)$:- $c(X_2, X_1, Y_1)$,
$d(Y_1, Z, Z_1), d(Y_2, Z_1, Z_2)$.

In our case, we propagate by again using a definition step and the folding rule. This results in the program below.

$p(X, Y, Z)$:- $e(X, Y, Z)$.
$p(X, Y, Z)$:- $c(X_1, X, Y), t(X_1, Z)$.
$t(X_1, Z)$:- $p(X_1, Y_1, Z_1), d(Y_1, Z, Z_1)$.

As above, we then transform the definition of t to obtain

$p(X, Y, Z)$:- $e(X, Y, Z)$.
$p(X, Y, Z)$:- $c(X_1, X, Y), t(X_1, Z)$.
$t(X_1, Z)$:- $e(X_1, Y_1, Z_1), d(Y_1, Z, Z_1)$.
$t(X_1, Z)$:- $c(X_2, X_1, Y_1), t(X_2, Z_1)$,
$d(Y_1, Z, Z_1)$.

Again, we may further optimise t by use of the magic set transformation. Note also that we may make the recursive call to t a bound query by moving the call to d before the recursive call to t.

The above examples show how the fold/unfold transformations may be used to propagate certain types of constraints. These same transformations have been used previously for constraint propagation [4], and as the latter examples show, it would seem that this method of propagation should be the standard method. It should also be noted that the techniques of [3] are specifically designed for linear recursive rules, whereas the fold/unfold transformations do not have this intrinsic limitation, and hence provide a more general framework for constraint propagation, which in turn may make use of specific properties, such as linear recursion.

5 Conclusion

We have seen how certain types of constraints may be propagated by the use of fold/unfold transformations. By using combinations of simple and well-understood transformations, we immediately get the desired correctness properties of the transformations, as well as conceptual simplicity. Hence, the main problem that remains to determine when a constraint exists, and what its nature is. In this paper we have mentioned magic sets as a way of propagating binding constraints, i.e. constraints such as $X = 1$, as well as more subtle constraints such as the constraints on Y implied by the goal $p(X, Y), q(Y, Z)$. Other forms of constraint may be used as well, such as filtering constraints, or integrity constraints. The important point is that once a constraint is identified, the propagation of the constraint should be straightforward, using known techniques. In the future we hope to explore the possibilities for such a methodology using integrity constraints for optimization, in the manner of Minker et al. [2]. Further work on existential queries also seems promising.

6 Acknowledgements

The authors would like to thank Isaac Balbin and David Kemp for valuable discussions of this material.

The support of the Australian Research Council through the Machine Intelligence Project and that of the Centre for Intelligent Decision Systems is gratefully acknowledged.

References

[1] P. Gardiner and J. Shepherdson, Unfold/Fold Transformations of Logic Programs, in *Computational Logic: Essays in Honour of Alan Robinson* 565-583, J-L. Lassez and G. Plotkin (eds.), MIT Press, 1991.

[2] J. Grant and J. Minker, Integrity Constraints in Knowledge Based Systems, Technical Report CS-TR-2223, Department of Computer Science, University of Maryland, 1989.

[3] J. Han, Is it Possible to Capture More Bindings than Magic Rule Rewriting?, Proceedings of the ILPS'91 workshop on Deductive Databases, San Diego, October, 1991.

[4] D. Kemp, K. Ramamohanrao, I. Balbin and K. Meenakshi, Propagating Constraints in Recursive Deductive Databases, *Proceedings of the North American Conference on Logic Programming* 981-998, Cleveland, October, 1989.

[5] J. Naughton, R. Ramakrishnan, Y. Sagiv and J. Ullman, Efficient Evaluation of Right-, Left-, and Multi-Linear Rules, *Proceedings of the SIGMOD International Symposium on Management of Data* 235-242, Portland, May, 1989.

[6] J. Naughton, R. Ramakrishnan, Y. Sagiv and J. Ullman, Argument Reduction by Factoring, *Proceedings of the Fifteenth International Conference on Very Large Data Bases* 173-182, Amsterdam, August, 1989.

[7] R. Ramakrishnan, C. Beeri and R. Krishnamurthy, Optimizing Existential Datalog Queries, *Proceedings of the ACM Symposium on Principles of Database Systems* 89-102, Austin, March, 1988.

[8] Y. Sagiv, Is There Anything Better than Magic?, *Proceedings of the North American Conference on Logic Programming* 235-254, Austin, October, 1990.

[9] T. Sato and H. Tamnaki, Deterministic Transformation and Determinitic Synthesis, *Programming of Future Genreation Computers II* 307-327, K. Fuchi and L. Koptt (eds.), North-Holland, 1988.

[10] D. Srivastava and R. Ramakrishnan, Fold/Unfold in Constraint Query Optimization, draft manuscript, 1991.

[11] H. Tamaki and T. Sato, Unfold/Fold Transformation of Logic Programs, *Proceedings of the Second International Conference on Logic Programming* 127-138, Uppsala, July, 1984.

[12] J. Ullman, Implementation of Logical Query Languages for Databases, *ACM Transactions on Database Systems* 10:3:289-321, 1985.