Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

K.R. Apt

Efficient computing of least fixpoints

Computer Science/Department of Software Technology Report CS-R8830 September

69F41, 69F32, 69H21

Efficient Computing of Least Fixpoints

Krzysztof R. Apt
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
and
Department of Computer Sciences,
University of Texas at Austin, Austin, Texas 78712-1188, U.S.A.

We systematically derive efficient algorithms for computing the least fixpoints of monotonic operators in the context of deductive databases. To this purpose we identify *additivity* as a key property assumed from the operator and study additive operators on complete lattices. To handle the general case certain combinations of additive operators need to be considered.

1980 CR categories: F.4.1, F.3.2, H.2.1.
1985 Mathematics Subject Classification: 68Q40, 68T99
Key Words & Phrases: deductive databases, additive operators, least fixpoints, non-redundancy.

1. Introduction

Bottom up computing lies at the heart of the deductive databases. It is an indispensable component in a bottom up evaluation of queries. It boils down to the problem of computing the closure of a finite set of facts under a finite set of rules in a first order language without function symbols.

In this paper we provide a systematic account of the efficient algorithms for computing this closure. To this purpose we reformulate this problem as the problem of computing the least fixpoint of a monotonic operator formed in a natural way from the given set of facts and set of rules.

This allows us to study the original problem in an abstract setting. We identify the general principles on which these algorithms rely by studying *additive* operators on complete lattices and their least fixpoints. Additivity is a stronger property than monotonicity and turns out to be a key concept in these investigations. A study of additive operators is sufficient for handling a special case of the problem.

To handle the general case we need to modify appropriately the framework and study certain combinations of additive operators. The correctness of derived algorithms is a simple consequence of certain general theorems concerning powers of additive operators or their combinations. We show that some of these algorithms are in a certain sense non-redundant.

The algorithms obtained, when reformulated in the original context of deductive databases become essentially the algorithms proposed in Rohmer, Lescoeur and Kerisit [RLK], Balbin and Ramamohanarao [BaR] and elsewhere.

2. Preliminaries

2.1. Closures

Throughout this paper we consider a first order language without function symbols. By a *fact* we mean a ground atom in this language and by a *clause* a formula in this language of the form

$$A_0 \leftarrow A_1, \ldots, A_m$$

where $m \geqslant 0$ and each A_i is an atom. Such a clause is to be read as "if A_1, ..., and A_n, then A_0". When $m = 0$ we call such a clause a *unit clause* and when $m \geqslant 1$ we call such a clause a *rule*.

Let now **A** be a finite non-empty set of facts and **R** a finite non-empty set of rules. By the closure

Report CS-R8830
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

of **A** under **R**, denoted by $Cl_{\mathbf{R}}(\mathbf{A})$, we mean the least set of facts containing **A** and closed under **R**. A set of facts **B** is *closed under* **R** if for any rule $A_0 \leftarrow A_1, \cdots, A_m$ from **R** the following closure condition holds:

$$\{A_1\theta, \ldots, A_m\theta\} \subseteq \mathbf{B} \Rightarrow A_0\theta \in \mathbf{B},$$

for every ground substitution θ defined on all variables of $A_0 \leftarrow A_1, \ldots, A_m$. A closure is easily seen to exist.

The problem of an efficient computing of the closure $Cl_R(\mathbf{A})$ is central in the theory of deductive databases. When **A** is the set of facts in the deductive database and **R** the set of rules, then $Cl_{\mathbf{R}}(\mathbf{A})$ is the intended meaning of this deductive database. While usually computation of the whole set $Cl_{\mathbf{R}}(\mathbf{A})$ is not practical, the computation of $Cl_{\mathbf{R}'}(\mathbf{A}')$ for suitably smaller sets **A**$'$ and **R**$'$ derived from **A** and **R**, turns out to be an essential ingredient of a bottom up evaluation of queries (see e.g. Rohmer, Lescoeur and Kerisit [RLK] and Beeri and Ramakrishnan [BR]).

The problem of an efficient computing of the closure $Cl_{\mathbf{R}}(\mathbf{A})$ has been extensively discussed in the literature (see e.g. Ullman [U] (Chapter 3) and Balbin and Ramamohanarao [BaR] where also previous references on this subject can be found).

For a subsequent study of closures we need to introduce some basic concepts concerning monotonic operators and their powers.

2.2. Powers of monotonic operators

Let L be a complete lattice with the least element L and ordering $\subseteq$. $X \cup Y$ stands for the *union* of elements X and Y of L.

An operator T on L is *monotonic* if for all X, Y

$$X \subseteq Y \Rightarrow T(X) \subseteq T(Y).$$

Given a monotonic operator T on L, we define its *powers* by putting for an element X of L

$$T\uparrow 0(X) = X,$$

$$T\uparrow(n+1)(X) = T(T\uparrow n(X)) \text{ for } n \geqslant 0,$$

and its *cumulative powers* by putting

$$T\Uparrow 0(X) = X,$$

$$T\Uparrow(n+1)(X) = T(T\Uparrow n(X)) \cup T\Uparrow n(X) \text{ for } n \geqslant 0.$$

We abbreviate $T\uparrow n(\emptyset)$ to $T\uparrow n$ and $T\Uparrow n(\emptyset)$ to $T\Uparrow n$. X is called a *fixpoint* of T if $T(X)=X$. X is *closed under* T if $T(X) \subseteq X$.

By the Knaster-Tarski theorem every monotonic operator T has the least fixpoint which we denote by *lfp*(T).

The following straightforward lemma summarizes the properties of powers and cumulative powers we shall need later.

LEMMA 1. *Let T be a monotonic operator.*
i) *For all $n \geqslant 0$ $T\uparrow n \subseteq T\uparrow(n+1)$.*
ii) *If for some n $T\uparrow n = T\uparrow(n+1)$ then for all $m > n$ $T\uparrow m = T\uparrow n$.*
iii) *For all X and $n \geqslant 0$ $T\Uparrow n(X) \subseteq T\Uparrow(n+1)(X)$.*
iv) *For all X, if for some n $T\Uparrow n(X) = T\Uparrow(n+1)(X)$, then for all $m > n$ $T\Uparrow m(X) = T\Uparrow n(X)$.*
v) *If L is finite then for some $n \geqslant 0$ lfp$(T) = T\uparrow n$.*
vi) *If L is finite then for some $n \geqslant 0$ $T\Uparrow n(X)$ is the least element of L containing X and closed under T.*
□

Given now an element Z of L we associate with T an operator $T[Z]$ defined by

$$T[Z](X) = T(X) \cup Z.$$

When T is monotonic, $T[Z]$ is monotonic, as well and its powers are related to the cumulative powers of T by the following lemma.

LEMMA 2. *Suppose that $T(\varnothing) = \varnothing$. Then for all Z and $n \geqslant 0$*

$$T[Z]\uparrow(n+1) = T\Uparrow n(Z).$$

PROOF. We prove by induction on n that for all $n \geqslant 0$

$$T\Uparrow n(Z) \subseteq T(T\Uparrow n(Z)) \cup Z. \tag{*}$$

The base case is obvious. Assume (*) holds for some $n \geqslant 0$. Then

$$\begin{array}{rcl} T\Uparrow(n+1)(Z) & = & T(T\Uparrow n(Z)) \cup T\Uparrow n(Z) \\ \text{(by the induction hypothesis)} & \subseteq & T(T\Uparrow n(Z)) \cup T(T\Uparrow n(Z)) \cup Z \\ & = & T(T\Uparrow n(Z)) \cup Z \\ \text{(by Lemma 1 iii) and monotonicity of } T\text{)} & \subseteq & T(T\Uparrow(n+1)(Z)) \cup Z, \end{array}$$

so (*) indeed holds for all $n \geqslant 0$.

We now prove the lemma by induction on n using (*). We have

$$\begin{array}{rl} T[Z]\uparrow 1 = & T[Z](\varnothing) \\ = & T(\varnothing) \cup Z \\ \text{(since } T(\varnothing) = \varnothing) = & Z \\ = & T\Uparrow 0(Z) \end{array}$$

which settles the base case. Suppose now that the claim holds for some $n \geqslant 0$. Then

$$\begin{array}{rcl} T[Z]\uparrow(n+2) & = & T[Z](T[Z]\uparrow(n+1)) \\ \text{(by the induction hypothesis)} & = & T[Z](T\Uparrow n(Z)) \\ & = & T(T\Uparrow n(Z)) \cup Z \\ \text{(by (*))} & = & T(T\Uparrow n(Z)) \cup T\Uparrow n(Z) \cup Z \\ & = & T\Uparrow(n+1)(Z) \cup Z \\ \text{(by Lemma 1 iii))} & = & T\Uparrow(n+1)(Z). \quad \square \end{array}$$

This lemma shows that when $T(\varnothing) = \varnothing$, in order to study the cumulative powers $T\Uparrow n(Z)$ of T it is sufficient to study the "usual" powers $T[Z]\uparrow n$ of $T[Z]$.

2.3. *The immediate consequence operator*

To relate these observations to our particular situation we need first to recall some notation and terminology.

By a *logic program* we mean a finite, non-empty set of clauses. As in the assumed language there are no function symbols, we consider here only a special case of logic programs sometimes called *datalogs* or *deductive databases*. By B_P we denote the *Herbrand base* of P, i.e. the set of all ground atoms in the language of P. By the assumption B_P is finite. By T_P we denote the *immediate consequence operator* introduced in Van Emden and Kowalski [VEK]. T_P is an operator on the complete lattice formed from the subsets of B_P ordered by inclusion and is defined as follows:

$$T_P(I) = \{A : \text{for some } B_1, \ldots, B_n \in I \; A \leftarrow B_1, \ldots, B_n \text{ is a ground instance of a clause from } P\}.$$

It is well known and straightforward to prove that T_P is monotonic.

Let now $\mathbf{A}$ be a finite, non-empty set of facts and $\mathbf{R}$ a finite, non-empty set of rules. It is an immediate consequence of the definitions that the closure $Cl_{\mathbf{R}}(\mathbf{A})$ is the least subset of B_P containing

A and closed under the operator $T_{\mathbf{R}}$.

The following summarizes the relevant properties of the immediate consequence operator.

LEMMA 3.
i) *For all subsets I of $B_{\mathbf{R} \cup \mathbf{A}}$*

$$T_{\mathbf{R}}[\mathrm{A}](I) = T_{\mathbf{R} \cup \mathrm{A}}(I).$$

ii) *For all $n \geq 0$*

$$T_{\mathbf{R} \cup \mathrm{A}} \uparrow (n+1) \;=\; T_R \Uparrow n(\mathrm{A}).$$

iii) $Cl_{\mathbf{R}}(\mathrm{A}) = \mathit{lfp}(T_{\mathbf{R} \cup \mathrm{A}})$.

PROOF. i) Clear, but note that $T_{\mathbf{R}}$ is considered here as an operator on a possibly larger lattice consisting of all subsets of $B_{\mathbf{R} \cup \mathrm{A}}$.
ii) By i) and Lemma 2. Note that $T_{\mathbf{R}}(\varnothing) = \varnothing$.
iii) By ii) and Lemma 1 ii), iv) v), vii). □

Of course iii) can also be proved directly without resorting to Lemma 2.

We thus see that the problem of computing the closure $Cl_{\mathbf{R}}(\mathrm{A})$ reduces to the problem of computing the least fixpoint of an operator T_P. This leads us to the study of least fixpoints of monotonic operators in a general setting. In this specific situation the appropriate level of abstraction is achieved when studying monotonic operators on products of complete lattices.

3. ADDITIVE OPERATORS

Let $L_1, \ldots, L_k, k \geq 1$, be fixed complete lattices, each with its least element $\varnothing$ and ordering $\subseteq$. From now on $L = L_1 \times \ldots \times L_k$ stands for the complete lattice whose domain is the Cartesian product of the domains of $L_1, \ldots, L_k$, and whose operations are interpreted componentwise. We use $\varnothing$ to denote the least element of L and $\subseteq$ to denote the ordering of the elements of L. The operations of $\cup$ (union) and $\setminus$ (difference) extend from $L_1, \ldots, L_k$ to L componentwise. From the context it will be always clear to which lattice $\varnothing$ belongs or on what lattice a particular operation is interpreted.

We denote the elements of L by X, Y, Z. $(X)_i$ stands for the i-th component of X and $X[i/Y]$ for the element obtained from X by replacing in it the ith component, $(X)_i$, by $(Y)_i$.

To every set A there corresponds a complete lattice $L(A)$ whose domain consists of all the subsets of A ordered by set inclusion. Later on we shall deal only with such complete lattices or their products.

Note that using this notation T_P is an operator on $L(B_P)$. The following notion will play a crucial role in our investigations.

DEFINITION 4. An operator T on L is *additive* if for all i, $1 \leq i \leq k$, X, Y and Z

$$T(X[i/Y \cup Z]) \;=\; T(X[i/Y]) \cup T(X[i/Z])$$

Thus an additive operator distributes union over all its components. Clearly, every additive operator is monotonic but the converse is not true.

EXAMPLE 5. Consider the complete lattice $L(\{0,1\})$ and the operator T on $L(\{0,1\})$ defined by

$$T(\varnothing) = \varnothing,$$

$$T(\{0\}) = T(\{1\}) = \{0\},$$

$$T(\{0,1\}) = \{0,1\}.$$

Then T is monotonic but not additive. □

By the definition of the union operation, the identity operator on L is additive. Other, more interesting additive operators will be studied in Sections 6 and 9.

The following lemma captures an important property of the additive operators.

LEMMA 6. *Suppose T is additive. Then for all X, Y such that $Y \subseteq X$*

$$T(X) = \bigcup_{i=1}^{k} T(X[i/X \setminus Y]) \cup T(Y)$$

This lemma states that the value of T on X can be computed in terms of values of T on some elements smaller than X.

PROOF. We have by additivity

$$\begin{aligned} T(X) &= T(Z) \text{ where for } j=1,...,k\ (Z)_j = (Y)_j \cup (X \setminus Y)_j \\ &= \cup\{T(Z)\text{: for } j=1,...,k\ (Z)_j = (Y)_j \text{ or } (Z)_j = (X \setminus Y)_j\} \\ &= \bigcup_{i=1}^{k}\{T(Z)\text{: } (Z)_i = (X \setminus Y)_i \text{ and for } j=1,...,k,\ j \neq i\ (Z)_j = (Y)_j \text{ or } (Z)_j = (X \setminus Y)_j\} \cup T(Y) \\ &= \bigcup_{i=1}^{k}\{T(Z)\text{: } (Z)_i = (X \setminus Y)_i \text{ and for } j=1,...,k,\ j \neq i\ (Z)_j = (Y)_j \cup (X \setminus Y)_j\} \cup T(Y) \\ &= \bigcup_{i=1}^{k} T(X[i/X \setminus Y]) \cup T(Y). \quad \square \end{aligned}$$

As the identity operator is additive, we obtain as a corollary the equality

$$X = \bigcup_{i=1}^{k} X[i/X \setminus Y] \cup Y$$

where $Y \subseteq X$.

When L is a product of k lattices, each of the form $L(A)$, then this equality corresponds to the logical equivalence

$$\begin{aligned} &(a_1, \ldots, a_k) \in X \leftrightarrow \exists i(1 \leqslant i \leqslant k \wedge a_i \in (X \setminus Y)_i \wedge \forall j(1 \leqslant j \leqslant k \wedge i \neq j \rightarrow a_j \in (X)_j)) \\ &\qquad \vee (a_1, \ldots, a_n) \in Y \text{ where } Y \subseteq X. \end{aligned}$$

Next lemma improves upon the above lemma by showing that the value of T on X can be computed in terms of values of T on some elements smaller than $X[i/X \setminus Y]$ and Y.

LEMMA 7. *Suppose T is additive. Then for all X, Y such that $Y \subseteq X$*

$$T(X) = \bigcup_{i=1}^{k} T(R(i, X, Y)) \cup T(Y)$$

where

$$R(i, X, Y) = X[1/Y]...[i-1/Y][i/X \setminus Y]$$

PROOF. Fix some $i, 1 \leqslant i \leqslant k$. Consider some Z where $(Z)_i = (X \setminus Y)_i$ and for $j=1,...,k,\ i \neq j$ $(Z)_j = (Y)_j$ or $(Z)_j = (X \setminus Y)_j$. Let j_o be the smallest j such that $(Z)_j = (X \setminus Y)_j$.

Then

- $j_o \leqslant i$,
- for all $j, 1 \leqslant j < j_o$ $(Z)_j = (Y)_j$,
- $(Z)_{j_o} = (X \setminus Y)_{j_o}$,
- for all $j, j_o < j \leqslant k$ $(Z)_j \subseteq (X)_j$.

Thus $Z \subseteq R(j_o,X,Y)$ and consequently

$$Z \subseteq \bigcup_{j=1}^{i} R(j,X,Y).$$

Now

$T(X)$

{as in the proof of Lemma 6}

$= \bigcup_{i=1}^{k} \{T(Z)\colon (Z)_i = (X \setminus Y)_i \text{ and for } j=1,\dots,k,\ j \neq i\ (Z)_j = (Y)_j \text{ or } (Z)_j = (X \setminus Y)_j\} \cup T(Y)$

{by monotonicity of T and (*)}

$\subseteq \bigcup_{i=1}^{k} \bigcup_{j=1}^{i} T(R(j,X,Y)) \cup T(Y)$

$= \bigcup_{i=1}^{k} T(R(i,X,Y)) \cup T(Y).$

To prove the inclusion in the other direction first note that for $i=1,\dots,k$

$$R(i,X,Y) \subseteq X[i/X \setminus Y].$$

Thus by the monotonicity of T for $i=1,\dots,k$

$$T(R(i,X,Y)) \subseteq T(X[i/X \setminus Y])$$

and the desired inclusion follows by Lemma 6. □

By the additivity of the identity operator, we obtain as a corollary the equality

$$X = \bigcup_{i=1}^{k} R(i,X,Y) \cup Y$$

where $Y \subseteq X$.

When L is a product of k lattices, each of the form $L(A)$, this equality corresponds to the following logical equivalence

$$\begin{aligned}(a_1,\dots,a_n) \in X \leftrightarrow\ & \exists i(1 \leqslant i \leqslant k \\ & \wedge \forall j(1 \leqslant j < i \rightarrow a_j \in (Y)_j) \\ & \wedge a_i \in (X \setminus Y)_i \\ & \wedge \forall j(i < j \leqslant k \rightarrow a_j \in (X)_j)) \\ & \vee (a_1,\dots,a_n) \in Y\end{aligned}$$

where $Y \subseteq X$.

Call now two elements of X, Y of L *disjoint* if for some i, $1 \leqslant i \leqslant k$

$$(X)_i \cap (Y)_i = \varnothing.$$

The following observation will be helpful in the next section.

NOTE 8. Suppose $Y \subseteq X$. Then for all $i,j=1,\dots,k$
- for $i \neq j$ $R(i,X,Y)$ and $R(j,X,Y)$ are disjoint,
- $R(i,X,Y)$ and Y are disjoint.

PROOF. Suppose $i<j$. Then $(R(j,X,Y))_i = (Y)_i$. But $(R(i,X,Y))_i = (X \setminus Y)_i$, so

$$(R(i,X,Y))_i \cap (R(j,X,Y))_i = \varnothing$$

and

$$(R(i,X,Y))_i \cap (Y)_i = \varnothing. \quad \square$$

The above note in conjunction with Lemma 7 states that for an additive operator $T, T(X)$ can be computed in terms of values of T on some elements smaller than X which are pairwise disjoint.

4. Computing powers of additive operators

Let T be a monotonic operator on L. We define

$$\Delta T\uparrow 0 = \varnothing,$$

$$\Delta T\uparrow(n+1) = T\uparrow(n+1) \setminus T\uparrow n \text{ for } n \geqslant 0.$$

As by Lemma 1 we have $T\uparrow n \subseteq T\uparrow(n+1)$ for $n \geqslant 0$, $\Delta T\uparrow n$ denotes the increase computed during the n-th iteration of T.

The following theorem shows how $T\uparrow(n+1)$ can be computed using $\Delta T\uparrow n$ when T is additive.

Theorem 9. *Suppose T is additive. Then for all $n \geqslant 0$*

$$T\uparrow(n+1) = \bigcup_{i=1}^{k} T(T\uparrow n[i/\Delta T\uparrow n]) \cup T\uparrow n.$$

Proof. For $n=0$ the claim clearly holds, so assume that $n>0$. We have

$$T\uparrow(n+1)$$

$$= T(T\uparrow n)$$

$\{$by Lemma 6 with $X := T\uparrow n$ and $Y := T\uparrow(n+1)\}$

$$= \bigcup_{i=1}^{k} T(T\uparrow n[i/\Delta T\uparrow n]) \cup T(T\uparrow(n-1))$$

$$= \bigcup_{i=1}^{k} T(T\uparrow n[i/\Delta T\uparrow n]) \cup T\uparrow n. \quad \square$$

We now show that $T\uparrow(n+1)$ can be computed in a different way which will lead to a more efficient algorithm. For the notational purposes we assume that $T\uparrow(-1) = \varnothing$.

Theorem 10. *Suppose T is additive. Then for all $n \geqslant 0$*

$$T\uparrow(n+1) = \bigcup_{i=1}^{k} T(S(i,n)) \cup T\uparrow n,$$

where

$$S(i,n) = T\uparrow n[1/T\uparrow(n-1)]...[i-1/T\uparrow n-1)][i/\Delta T\uparrow n].$$

Thus using the notation of Section 3,

$$S(i,n) = R(i, T\uparrow n,\ T\uparrow(n-1)). \tag{1}$$

Note that for $i = 1,...,k$

$$S(i,n) \subseteq T\uparrow n[i/\Delta T\uparrow n],$$

so $T\uparrow(n+1)$ is now computed in terms of smaller values than those given in Theorem 9.

Proof. We leave to the readers checking that the theorem holds for $n=0$. Assume $n>0$. We have

$$T\uparrow(n+1)$$
$$= T(T\uparrow n)$$

{by Lemma 7 with $X:=T\uparrow n$ and $Y:=T\uparrow(n-1)$}

$$= \bigcup_{i=1}^{k} T(R(i,T\uparrow n,\ T\uparrow(n-1))\ \cup\ T(T\uparrow(n-1))$$

{since $R(i,\ T\uparrow n,\ T\uparrow(n-1))=S(i,n)$}

$$= \bigcup_{i=1}^{k} T(S(i,n))\ \cup\ T\uparrow n.\quad \Box$$

We now show that the arguments of T used in the computation of the sequence $T\uparrow n$, $n=0,1,...$ by means of the method given in Theorem 10 are pairwise disjoint.

Lemma 11. *For all i,j,m,n such that $m,n\geqslant 0$, $1\leqslant i,j\leqslant k$, if $(i,m)\neq(j,n)$ then $S(i,m)$ and $S(j,n)$ are disjoint.*

Proof. Suppose $i\neq j$ and $m=n$. Then by (1) and Note 8 $S(i,m)$ and $S(j,n)$ are disjoint.

Suppose now that $m\neq n$, say $m<n$. Then, again by (1) and Note 8, $S(j,n)$ and $T\uparrow(n-1)$ are disjoint. But

$$S(i,m)\ \subseteq\ T\uparrow m\ \subseteq\ T\uparrow(n-1),$$

so $S(i,m)$ and $S(j,n)$ are disjoint, as well. $\Box$

5. Algorithms I

We now apply the results obtained in the previous section to derive efficient algorithms computing the least fixpoint of an additive operator. The following simple observations will be used in the sequel.

Note 12. For a monotonic operator T on L, for all $n\geqslant 0$

i)

$$T\uparrow(n+1)\ =\ T\uparrow n\ \cup\ \Delta T\uparrow(n+1),$$

ii) if $\Delta T\uparrow n=\varnothing$ then $T\uparrow n=lfp(T)$. $\Box$

We now define for an additive operator T on L

$$\Delta\Delta T\uparrow 0\ =\ \varnothing,$$

$$\Delta\Delta T\uparrow(n+1)\ =\ \bigcup_{i=1}^{k} T(T\uparrow n[i/\Delta T\uparrow n]).$$

Note 13. For an additive operator T on L, for all $n\geqslant 0$

$$\Delta T\uparrow(n+1)\ =\ \Delta\Delta T\uparrow(n+1)\setminus T\uparrow n.$$

Proof. By Theorem 9. $\Box$

Assume now that the lattice L is finite and consider a monotonic operator T on L. By Lemma 1 eventually $T\uparrow n=T\uparrow(n+1)$, i.e. eventually $\Delta T\uparrow n=\varnothing$ and then by Note 12 $T\uparrow n$ equals the least fixpoint $lfp(T)$ of T.

The following simple minded algorithm computes the least fixpoint of T using Note 9. We introduce here a variable t to compute $T\uparrow n$ and a variable Δt to compute $\Delta T\uparrow n$. Here and in the subsequent algorithms we use the assignments to an auxiliary variable n to express the relevant invariants and assertions. The assignments to n do not belong to the algorithm. Formally, this use of auxiliary variables to establish correctness of an algorithm is justified by the rule of auxiliary variables of Owicki and Gries [OG].

Algorithm 1

```
{$n:=0;$}
$t:=\varnothing$;
$\Delta t:=\varnothing$;
repeat {$t=T\uparrow n \wedge \Delta t=\Delta T\uparrow n$}
   $\Delta t:=T(t)\setminus t$;
   $t:=t\cup\Delta t$;
   {$n:=n+1$}
until $\Delta t=\varnothing$
{$t=T\uparrow n \wedge \Delta T\uparrow n=\varnothing$}
{$t=lfp(T)$}
```

When T is additive we can use Theorem 9 to compute its least fixpoint more efficiently. To this purpose we additionally introduce a variable $\Delta\Delta t$ to compute $\Delta\Delta T\uparrow n$. This algorithm relies on Notes 12 and 13.

Algorithm 2

```
{$n:=0;$}
$t:=\varnothing$;
$\Delta t:=\varnothing$;
$\Delta\Delta t:=\varnothing$;
repeat {$t=T\uparrow n \wedge \Delta t=\Delta T\uparrow n$}
   $i:=1$;
   while $i\neq k+1$ do {$\Delta\Delta t=\bigcup_{j=1}^{i-1} T(T\uparrow n[i/\Delta T\uparrow n])$}
      $\Delta\Delta t:=\Delta\Delta t\cup T(t[i/\Delta t])$
      $i:=i+1$
   od; {$\Delta\Delta t=\Delta\Delta T\uparrow(n+1)$}
   $\Delta t:=\Delta\Delta t\setminus t$;
   $t:=t\cup\Delta t$;
   $\Delta\Delta t:=\varnothing$;
   {$n:=n+1$}
until $\Delta t=\varnothing$
{$t=T\uparrow n \wedge \Delta T\uparrow n=\varnothing$}
{$t=lfp(T)$}
```

A more efficient version is obtained when using Theorem 10. To this purpose we now use the variable $\Delta\Delta t$ to compute $\bigcup_{i=1}^{k} T(S(i,n))$. Additionally, we introduce a variable s to maintain $T\uparrow(n-1)$ when computing $T\uparrow(n+1)$. The algorithm relies on the following observation.

NOTE 14. For an additive operator T on L, for all $n\geqslant 0$

$$\Delta T\uparrow(n+1) = \bigcup_{i=1}^{k} T(S(i,n)) \setminus T\uparrow n.$$

PROOF. By Theorem 10. □

Algorithm 3

```
{$n:=0$};
$s:=\emptyset$;
$t:=\emptyset$;
$\Delta t:=\emptyset$;
$\Delta\Delta t:=\emptyset$,
**repeat** {$s=T\uparrow(n-1)\wedge t=T\uparrow n\wedge\Delta t=\Delta T\uparrow n$}
    $i=1$;
    **while** $i\neq k+1$ **do** {$\Delta\Delta t=\bigcup_{j=1}^{i-1}T(S(i,n))$}
        $\Delta\Delta t:=\Delta\Delta t\cup T(t[1/s]...[i-1/s][i/\Delta t])$
        $i:=i+1$
    **od**;{$\Delta\Delta t = \bigcup_{i=1}^{k}T(S(i,n))$}
    $s:=t$;
    $\Delta t:=\Delta\Delta t\setminus t$;
    $t:=t\cup\Delta t$;
    $\Delta\Delta t:=\emptyset$;
    {$n:=n+1$}
**until** $\Delta t=\emptyset$
{$t=T\uparrow n\wedge\Delta T\uparrow n=\emptyset$}
{$t=lfp(T)$}
```

To avoid repeated computing of expressions of the form $t[1/s]...[i-1/s][i/\Delta t]$ used in the inner loop, we introduce a variable u to maintain this expression. This leads to the following improved version of the previous algorithm.

Algorithm 4

```
{$n:=0$};
$s:=\emptyset$;
$t:=\emptyset$;
$\Delta t:=\emptyset$;
$\Delta\Delta t:=\emptyset$;
**repeat** {$s=T\uparrow(n-1)\wedge t=T\uparrow n\wedge\Delta t=\Delta T\uparrow n$}
    $i:=1$;
    $u:=t[i/\Delta t]$;
    **while** $i\neq k+1$ **do** {$\Delta\Delta t=\bigcup_{j=1}^{i-1}T(S(j,n))\wedge u=S(i,n)$}
        $\Delta\Delta t:=\Delta\Delta t\cup T(u)$;
        $u:=u[i/s][i+1/\Delta t]$;
        $i:=i+1$
    **od**;{$\Delta\Delta t=\bigcup_{j=1}^{k}T(S(j,n))$}
    $s:=t$;
    $\Delta t:=\Delta\Delta t\setminus t$;
    $t:=t\cup\Delta t$;
    $\Delta\Delta t := \emptyset$;
    $\{n := n+1\}$
**until** $\Delta t = \emptyset$
$\{t = T\uparrow n \wedge \Delta T\uparrow n = \emptyset\}$
$\{t = lfp(T)\}$
```

6. Computing least fixpoints in deductive databases I

To apply these algorithms to deductive databases consider a logic program P. Let $p_1, \ldots, p_m$ be all relation symbols appearing in P. For a relation symbol r appearing in P denote by $[r]$ the set of all ground atoms in the Herbrand universe of P whose relation symbol is r.

Subsets of $[r]$ form a complete lattice $L([r])$. Denote $L([p_i])$ by L_i and, as in Sections 2-4 $L_1 \times ... \times L_m$ by L. Clearly L is isomorphic with $L(B_P)$, with an element $(I_1, \ldots, I_m)$ of L mapped to the element $\bigcup_{i=1}^{m} I_i$ of $L(B_P)$. Thus the immediate consequence operator T_P can be viewed as an operator on L.

To compute the least fixpoint of T_P we would like to use the algorithms developed in the previous section. Unfortunately, under the above interpretation, T_P is not additive.

Example 15. Consider a program P consisting of a single clause $p_2 \leftarrow p_1(1), p_1(2)$. Then

$$L = L([p_1]) \times L([p_2]) \text{ and}$$
$$T_P(\{p_1(1)\}, \emptyset) = \emptyset,$$
$$T_P(\{p_1(2)\}, \emptyset) = \emptyset$$

whereas

$$T_P(\{p_1(1), p_1(2)\}, \emptyset) = \{p_2\}.$$

Thus T_P is not additive. □

However, under certain natural restriction T_P is additive.

Definition 16. We call a program P *normal* when no relation symbol occurs twice in a hypothesis of a clause from P.

Theorem 17. *For a normal program P, T_P is additive.*

Proof. Consider a union $T_1 \cup T_2$ of two operators T_1 and T_2 on L defined by

$$T_1 \cup T_2(X) = T_1(X) \cup T_2(X).$$

Note that a union of two additive operators is additive and that for a program $P_1 \cup P_2$

$$T_{P_1 \cup P_2} = T_{P_1} \cup T_{P_2}.$$

Moreover when $P_1 \cup P_2$ is normal, then both P_1 and P_2 are normal.

So it suffices to prove that for a normal program P consisting of a single clause, T_P is additive. The proof is straightforward and left to the reader. □

Thus for a normal program P we may use the algorithms given in the previous section to compute the least fixpoint of T_P. However, to consider the general case we have to modify appropriately our theory. To this purpose we shall need to consider in Section 8 a more complex situation when the operator of interest is defined as an appropriate composition of additive operators.

7. Optimizations and non-redundancy I

The algorithms we presented in Section 5 can be somewhat improved when some information about the form of the operator T is available. We introduce the following notion.

Definition 18. An operator T on L is *strict* if for all i, $1 \leqslant i \leqslant k$ and X

$$T(X[i/\varnothing]) = \varnothing.$$

Thus a strict operator yields the least element when applied to an argument with a component consisting of the least element.

Suppose now that for some additive and strict operator V and an element Z, $T = V[Z]$, i.e. for all X

$$T(X) = V(X) \cup Z.$$

Then we can compute least fixpoint of T more efficiently by avoiding the repeated generation of Z during the computation of the powers of T. Instead Z can be generated once - at the beginning of the computation. This form of computing amounts to a computation of the cumulative powers of V starting at Z. Formally, this relationship is expressed in Lemma 2 which applies here as by strictness $V(\varnothing) = \varnothing$.

Assuming the above form of T, in case of Algorithm 2 we obtain the following improvement:

Algorithm 5

```
{$n := 1;$}
$t := Z;$
$\Delta t := Z;$
$\Delta\Delta t := Z;$
**repeat** {$t = T\uparrow n \wedge \Delta t = \Delta T\uparrow n$}
    $i := 1;$
    **while** $i \neq k+1$ **do** {$\Delta\Delta t = \bigcup_{j=1}^{i-1} T(T\uparrow n[i/\Delta T\uparrow n])$}
      $\Delta\Delta t := \Delta\Delta t \cup V(t[i/\Delta t])$
      $i := i+1$
    **od**; {$\Delta\Delta t = \Delta\Delta T\uparrow(n+1)$}
    $\Delta t := \Delta\Delta t \setminus t;$
    $t := t \cup \Delta t;$
    $\Delta\Delta t := Z;$
    {$n := n+1$}
**until** $\Delta t = \varnothing$
{$t = T\uparrow n \wedge \Delta T\uparrow n = \varnothing$}
{$t = lfp(T)$}
```

This algorithm can be justified by relying on the correctness of Algorithm 2 and on the form of T. Note that by the strictness of V we have $T\uparrow 1 = Z$.

Clearly it is superflous to apply a strict operator to arguments with a component equal $\varnothing$. We now modify the above algorithm so that such applications of V do not arise. To this purpose it suffices to replace the assignment

$$\Delta\Delta t = \Delta\Delta t \cup V(t[i/\Delta t])$$

by

$$\textbf{if } \forall j (1 \leqslant j \leqslant k \rightarrow (t[i/\Delta t])_j \neq \varnothing) \textbf{ then } \Delta\Delta t := \Delta\Delta \cup V(t[i/\Delta t]).$$

Some obvious improvements are possible here and left to the reader. They involve computing the sets of coordinates i for which $(t)_i = \varnothing$ and for which $(\Delta t)_i \neq \varnothing$ in front of the **while** loop.

The same modifications can be applied to Algorithms 3 and 4. In case of Algorithm 4 we obtain the following

Algorithm 6

```
{$n := 1;$}
$s := \varnothing;$
$t := Z;$
$\Delta t := Z;$
$\Delta\Delta t := Z;$
repeat {$s = T\uparrow(n-1) \wedge t = T\uparrow n \wedge \Delta t = \Delta t\uparrow n$}
    $i := 1;$
    $u := t[i/\Delta t];$
    while $i \neq k+1$ do {$\Delta\Delta t = \bigcup_{j=1}^{i-1} T(S(j,n)) \wedge u = S(i,n)$}
      if $\forall j(1 \leq j \leq k \rightarrow (u)_j \neq \varnothing)$ then $\Delta\Delta t := \Delta\Delta t \cup V(u);$
      $u := u[i/s][i+1/\Delta t];$
      $i := i+1;$
    od; {$\Delta\Delta t = \bigcup_{j=1}^{k} T(S(j,n))$}
    $s := t;$
    $\Delta t := \Delta\Delta t \setminus t;$
    $t := t \cup \Delta t;$
    $\Delta\Delta t := Z$
    {$n := n+1$}
until $\Delta t = \varnothing$
{$t = T\uparrow n \wedge \Delta T\uparrow n = \varnothing$}
{$t = lfp(T)$}
```

Also here some obvious improvements are possible. For example, during the first iteration through the **repeat** loop $s = \varnothing$, so the **while** loop, can be terminated after the first iteration.

We would like now to show that the last algorithm is in some sense non-repetitive, that is, it leads to a computation in which the operator V is continuously applied to "new" arguments.

To make this idea more precise we need to assume that each lattice L_i is of the form $L(A_i)$ for some finite set A_i. We have the following lemma.

LEMMA 19. *Suppose an operator T on L is additive and for all i, $1 \leq i \leq k$, $(X)_i \neq \varnothing$. Then*

$$T(X) = \bigcup\{T(\{a\}) : a \in X\}.$$

PROOF. Suppose that for some Y and some i, $1 \leq i \leq k$, $(Y)_i = \{a_1, \ldots, a_m\}$, where $m \geq 1$. Then by additivity

$$T(Y) = \bigcup_{j=1}^{m} T(Y[i/\{a_j\}]).$$

Using this formula m times we obtain the desired conclusion. □

This lemma shows that an additive operator when applied to arguments with all components different from $\varnothing$ is determined by its values on singletons.

Consider now Algorithm 6. Because of the conditonal assignment introduced within the **while** loop, the operator V is applied only to arguments with all components different from $\emptyset$. This means that in an execution of Algorithm 6 repetitions might arise if V were applied to two elements of L which, in the sense of Section 2, are not disjoint. Indeed, using the formula stated in Lemma 19, V would be then applied at least twice to the same singleton arguments.

It is easy to show that such repetitions cannot arise here. To this purpose it suffices to note that in Algorithm 6. V is applied exclusively to the arguments of the form $S(i,n)$, each time for a different pair (i,n). The desired conclusion now follows from Lemma 19.

When trying to apply the above findings to deductive databases we have to exercise some care. It is easy to see that $T_P = T_{\mathbf{R}}[\mathrm{A}]$, where

$$\mathbf{R} = \{C : C \text{ is a rule, } C \in P\}$$

$$\mathbf{A} = \{A : A \text{ is a ground instance of a unit clause from } P\}.$$

When $\mathbf{R}$ is normal, by Theorem 17, $T_{\mathbf{R}}$ is additive. However, $T_{\mathbf{R}}$ does not need to be strict, even for a normal $\mathbf{R}$.

Example 20. Consider a program P consisting of a single clause $p_2 \leftarrow p_1(1)$. Then P is normal but T_P is not strict. Indeed, for the element $(\{p_1(1)\}, \emptyset)$ of $L([p_1]) \times L([p_2])$ we have

$$T_P(\{p_1(1)\}, \emptyset) = \{p_2\}. \ \square$$

Thus, even for normal programs P we cannot use Algorithms 5 and 6. It is possible to restrict the class of considered programs even further to ensure strictness.

Definition 21. We call a program P *rich* if in every hypothesis of a clause from P all relation symbols of P occur.

Theorem 22. *For a rich program P, T_P is strict.*

Proof. Straightforward. $\square$

Thus when the set of rules occurring in P is normal and rich, we can apply Algorithms 5 and 6 to compute the least fixpoint of P. We now consider a more general set up which will allow us to handle arbitrary programs.

8. Computing powers of combinations of additive operators

Consider now $(m \geq 1)$ operators $T_1, \ldots, T_m$ such that for $i = 1,\ldots,m$

$$T_i : \mathbf{L}_i \to L, \qquad (*)$$

where

$$\mathbf{L}_i = L_{i,1} \times \ldots \times L_{i,k_i}$$

with each $L_{i,j}$ being equal to some $L_{f(i,j)}$ where $1 \leq f(i,j) \leq k$ and $k_i \geq 0$. When $k_i = 0$, T_i becomes a constant.

Each T_i induces an operator $\mathbf{T}_i$ on L. To define $\mathbf{T}_i$ we first associate with each element X from L an element $\{X\}_i$ from L_i by putting

$$\{X\}_i = ((X)_{i,1}, \ldots, (X)_{i,k_i}),$$

where $(X)_{i,j}$ stands for $(X)_{f(i,j)}$. We now define

$$\mathbf{T}_i(X) = T_i(\{X\}_i).$$

Suppose now that an operator T on L is defined as a union of the operators $\mathbf{T}_1, \ldots, \mathbf{T}_m$, i.e. suppose that for all X from L

$$T(X) = \bigcup_{i=1}^{m} \mathbf{T}_i(X). \tag{**}$$

We are interested in computing the least fixpoint of T when each operator T_i is additive. (It is clear how to extend the definition of additivity to the operators of the form of T_i). Note that then T is monotonic.

To this purpose we prove results analogous to Theorems 9 and 10. In what follows we assume that T is defined by (**) where each operator T_i is of the form (*).

THEOREM 23. *Suppose that each T_i for $i=1,...,m$ is additive. Then for all $n \geqslant 0$*

$$T\uparrow(n+1) = \bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{T\uparrow n\}_i[j/\{\Delta T\uparrow n\}_i]) \cup T\uparrow n.$$

As the notation becomes by now cryptic let us explain the meaning of the argument

$$\{T\uparrow n\}_i[j/\{\Delta T\uparrow n\}_i]$$

of T_i. It is obtained by replacing in the element

$$((T\uparrow n)_{i,1}, \ldots, (T\uparrow n)_{i,k_i})$$

of L_i the j-th component by $(\{\Delta T\uparrow n\}_i)_j$. Note that $(\{\Delta T\uparrow n\}_i)_j$ equals $(\Delta T\uparrow n)_{i,j}$.

PROOF. For $n=0$ both $T\uparrow n = \varnothing$ and $\Delta T\uparrow n = \varnothing$, so

$$\bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{T\uparrow 0\}_i[j/\{\Delta T\uparrow 0\}_i]) \cup T\uparrow 0$$

$$= \bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{\varnothing\}_i)$$

$$= \bigcup_{i=1}^{m} \mathbf{T}_i(\varnothing)$$

{by(**)}

$$= T(\varnothing)$$

$$= T\uparrow 1.$$

Assume now that $n>0$. We have

$$T\uparrow(n+1)$$

$$= T(T\uparrow n)$$

{by(**)}

$$= \bigcup_{i=1}^{m} \mathbf{T}_i(T\uparrow n)$$

$$= \bigcup_{i=1}^{m} T_i(\{T\uparrow n\}_i)$$

{by Lemma 3 with $T := T_i$, $X := \{T\uparrow n\}_i$ and $Y := \{T\uparrow(n-1)\}_i$}

$$= \bigcup_{i=1}^{m}(\bigcup_{j=1}^{k_i} T_i(\{T\uparrow n\}_i[j/\{\Delta T\uparrow n\}_i]) \cup T_i(\{T\uparrow(n-1)\}_i))$$

$$= \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(\{T\uparrow n\}_i[j/\{\Delta T\uparrow n\}_i]) \cup \bigcup_{i=1}^{m} \mathbf{T}_i(T\uparrow(n-1))$$

{by(**)}

$$= \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(\{T\uparrow n\}_i[j/\{\Delta T\uparrow n\}_i]) \cup T\uparrow n. \quad \square$$

THEOREM 24. Suppose that each T_i for $i=1,...,m$ is additive. Then for all $n \geqslant 0$

$$T\uparrow(n+1) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(U(i,j,n)) \cup T\uparrow n,$$

where for $i=1,...,m$ and $j=1,...,k_i$

$$U(i,j,n) = \{T\uparrow n\}_i[1/\{T\uparrow(n-1)\}_i]...[j-1/\{T\uparrow(n-1)\}_i][j/\{\Delta T\uparrow n\}_i].$$

It is helpful to note that $U(i,j,n)$ is obtained from the element $\{T\uparrow n\}_i$ of L_i by replacing the first component by $(T\uparrow(n-1))_{i,1}, \ldots,$ the $j-1$st component by $T\uparrow(n-1))_{i,j-1}$ and the j-th component by $(\Delta T\uparrow n)_{i,j}$. Thus, using the notation of Section 2,

$$U(i,j,n) = R(j, \{T\uparrow n\}_i, \{T\uparrow(n-1)\}_i). \qquad (***)$$

PROOF. It is straightforward to check that for $n=0$ the theorem holds as for $n=0$ $U(i,j,n)=\{\varnothing\}_i$. Suppose that $n>0$ we have

$$T\uparrow(n+1)$$

$$= T(T\uparrow n)$$

{by(**)}

$$= \bigcup_{i=1}^{m} \mathbf{T}_i(T\uparrow n)$$

$$= \bigcup_{i=1}^{m} T_i(\{T\uparrow n\}_i)$$

{by Lemma 4 with $T:=T_i$, $X:=\{T\uparrow n\}_i$ and $Y:=T\uparrow(n-1)\}_i$}

$$= \bigcup_{i=1}^{m} (\bigcup_{j=1}^{k_i} T_i(U(i,j,n)) \cup T_i(\{T\uparrow(n-1)\}_i))$$

{by (**), as in the proof of Theorem 18}

$$= \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(U(i,j,n)) \cup T\uparrow n. \quad \square$$

We now show that for each $i=1,...,m$, the arguments of T_i used in the computation of the sequence $T\uparrow n$, $n=0,1,...$ by means of the method given in Theorem 24 are pairwise disjoint.

LEMMA 25. *For all i, j_1, j_2, n_1, n_2 such that $1 \leqslant i \leqslant m$, $n_1, n_2 \geqslant 0$, and $1 \leqslant j_1, j_2 \leqslant k_i$, if $(j_1,n_1) \neq (j_2,n_2)$ then $U(i,j_1,n_1)$ and $U(i,j_2,n_2)$ are disjoint.*

PROOF. Suppose $j_1 \neq j_2$ and $n_1 = n_2$. Then by (***) and Note 8 $U(i,j_1,n_1)$ and $U(i,j_2,n_2)$ are disjoint.

Suppose now that $n_1 \neq n_2$, say $n_1 < n_2$. Then again by (***) and Note 8, $U(i,j_2,n_2)$ and $\{T\uparrow(n_2-1)\}_i$ are disjoint. But

$$U(i,j_1,n_1) \subseteq \{T\uparrow n_1\}_i \subseteq \{T\uparrow(n_2-1)\}_i$$

so $U(i,j_1,n_1)$ and $U(i,j_2,n_2)$ are disjoint, as well. □

9. Algorithms II

We now apply the results obtained in the previous section to derive efficient algorithms computing the least fixpoint of an operator T on L defined by (**).

We assume that the lattice L is finite. The first algorithm is obtained by using Theorem 23. We introduce a variable t to compute $T{\uparrow}n$, a variable Δt to compute $\Delta T{\uparrow}n$, and a variable $\Delta\Delta t$ to compute

$$\bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{T{\uparrow}n\}_i[j/\{\Delta T{\uparrow}n\}_i].$$

We use the following observation.

Note 26. Suppose that each T_i for $1=1,...,m$ is additive. Then for all $n \geqslant 0$

$$\Delta T{\uparrow}(n+1) = \bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{T{\uparrow}n\}_i[j/\{\Delta T{\uparrow}n\}_i] \setminus T{\uparrow}n.$$

Proof. By Theorem 23. □

Algorithm 7

```
{$n:=0;$}
$t:=\varnothing;$
$\Delta t:=\varnothing;$
$\Delta\Delta t:=\varnothing;$
repeat {$t=T{\uparrow}n \wedge \Delta t = \Delta T{\uparrow}n$}
    $i:=1;$
    while $i \neq m+1$ do {$\Delta\Delta t = \bigcup_{l=1}^{i-1}\bigcup_{j=1}^{k_l} T_l(\{T{\uparrow}n\}_l[j/\{\Delta T{\uparrow}n\}_l]$}
        $j:=1;$
        while $j \neq k_i+1$ do
            $\Delta\Delta t:=\Delta\Delta t \cup T_i\ (\{t\}_i[j/\{\Delta t\}_i]);$
            $j:=j+1$
        od;
        $i:=i+1$
    od; {$\Delta\Delta t = \bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(\{T{\uparrow}n\}_i[j/\{\Delta T{\uparrow}n\}_i]$}
    $\Delta t:=\Delta\Delta t \setminus t;$
    $t:=t \cup \Delta t;$
    $\Delta\Delta t:=\varnothing;$
    {$n:=n+1$}
until $\Delta t=\varnothing$
{$t=T{\uparrow}n \wedge \Delta T{\uparrow}n=\varnothing$}
{$t=lfp(T)$}
```

The next algorithm is obtained by using Theorem 24. We now use the variable $\Delta\Delta t$ to compute $\bigcup_{i=1}^{m}\bigcup_{j=1}^{k_i} T_i(U(i,j,n))$ and introduce a variable s to maintain $T{\uparrow}(n-1)$ while computing $T{\uparrow}(n+1)$. We use the following observation.

Note 27. Suppose that each T_i for $i=1,...,m$ is additive. Then for all $n \geqslant 0$

$$\Delta T\uparrow(n+1) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(U(i,j,n)) \setminus T\uparrow n.$$

PROOF. By Theorem 19. $\square$

Algorithm 8

```
{$n := 0$};
$s := \varnothing$;
$t := \varnothing$;
$\Delta t := \varnothing$;
$\Delta\Delta t := \varnothing$;
repeat {$s = T\uparrow(n-1) \wedge t = T\uparrow n \wedge \Delta t = \Delta T\uparrow n$}
    $i := 1$;
    while $i \neq m+1$ do {$\Delta\Delta t = \bigcup_{l=1}^{i-1} \bigcup_{j=1}^{k_l} T_l(U(l,j,n))$}
        $j := 1$;
        while $j \neq k_i + 1$ do
            $\Delta\Delta t := \Delta\Delta t \cup T_i\ (\{t\}_i[1/\{s\}_i]\ldots[j-1/\{s\}_i]\,[j/\{\Delta t\}_i])$;
            $j := j+1$
        od;
        $i := i+1$
    od; {$\Delta\Delta t = \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(U(i,j,n))$}
    $s := t$;
    $\Delta t := \Delta\Delta t \setminus t$;
    $t := t \cup \Delta t$;
    $\Delta\Delta t := \varnothing$;
    {$n := n+1$}
until $\Delta t = \varnothing$
{$t = T\uparrow n \wedge \Delta T\uparrow n = \varnothing$}
{$t = lfp(T)$}
```

Analogously as in Section 5, to avoid repeated computing of expressions of the form $\{t\}_i[1/\{s\}_i]\ldots[j-1/\{s\}_i][j/\{\Delta t\}_i]$ used in the inner loop, we introduce an array variable u to maintain in each $u[i]$ for $i = 1,\ldots,m$ the above expression. This leads to the following improvement.

Algorithm 9

```
{$n := 0$};
$s := \varnothing$;
$t := \varnothing$;
$\Delta t := \varnothing$;
$\Delta\Delta t := \varnothing$;
repeat {$s = T\uparrow(n-1) \wedge t = T\uparrow n \wedge \Delta t = \Delta T\uparrow n$}
    $i := 1$;
    while $i \neq m+1$ do {$\Delta\Delta t = \bigcup_{l=1}^{i-1} \bigcup_{j=1}^{k_l} T_l(U(l,j,n))$}
        $j := 1$;
        $u[i] := \{t\}_i[j/\{\Delta t\}_i]$;
        while $j \neq k_i + 1$ do {$u[i] = U(i,j,n)$}
            $\Delta\Delta t := \Delta\Delta t \cup T_i(u[i])$;
            $u[i] := u[i][j/\{s\}_i][j+1/\{\Delta t\}_i]$;
            $j:=j+1$
        od;
        $i:=i+1$
    od; $\{\Delta\Delta t = \bigcup_{i=1}^{m} \bigcup_{j=1}^{k_i} T_i(U(i,j,n))\}$
    $s:=t$;
    $\Delta t:=\Delta\Delta t \setminus t$;
    $t:=t \cup \Delta t$;
    $\Delta\Delta t:=\varnothing$;
    $\{n:=n+1\}$
until $\Delta t=\varnothing$
$\{t=T\uparrow n \wedge \Delta T\uparrow n=\varnothing\}$
$\{t=lfp(T)\}$
```

10. Computing least fixpoints in deductive databases II

We now show that we can apply the algorithms developed in the previous section to deductive databases. As opposed to Section 6 we consider here an arbitrary case. Assume a program P with relation symbols $p_1, \ldots, p_k$ and adopt the notation of Section 6. Consider a clause C of P of the form

$$A \leftarrow A_1, \ldots, A_n.$$

Let $r_1, \ldots, r_n$ be the relation symbols appearing in $A_1, \ldots, A_n$, respectively; $r_1, \ldots, r_n$ do not need to be pairwise different. Each r_i equals some p_j, say $p_{f(i)}$.

The clause C induces an operator T_C from $L_{f(1)} \times \ldots \times L_{f(n)}$ into L defined as follows:

$$T_C(I_1, \ldots, I_n) = \{B\text{: for some } B_1 \in I_1, \ldots, B_n \in I_n,\ B \leftarrow B_1, \ldots, B_n \text{ is a ground instance of } C\}.$$

The following observation will be crucial in the sequel.

Note 28. For each clause C the operator T_C is additive.

Proof. Straightforward. □

Given now a program $P=\{C_1, \ldots, C_m\}$ consider the sequence of operators $T_{C_1}, \ldots, T_{C_m}$ from appropriate product lattices into L. As in Section 7 each operator T_{C_i} induces an operator T_{C_i} on L. The following lemma relates the immediate consequence operator T_P to the operators $T_{C_1}, \ldots, T_{C_m}$.

Lemma 29. *For a program* $P=\{C_1, \ldots, C_m\}$

$$T_P(I) = \bigcup_{i=1}^{m} \mathrm{T}_{C_i}(I)$$

for every Herbrand interpretation I *of* P.

Proof. It suffices to observe that for $i=1,\ldots,m$

$\mathrm{T}_{C_i}(I) = \{B\text{: for some } B_1, \ldots, B_n \in I,\ B \leftarrow B_1, \ldots, B_n \text{ is a ground instance of } C_i\}$. □

This means that for a program P we can compute the least fixpoint of T_P using the algorithms given in the last section.

Algorithm 7 then becomes essentially the algorithm proposed in Rohmer, Lescoeur and Kerisit [RLK] (see also Ullman [U] (Chapter 3)), whereas Algorithm 8 becomes essentially the algorithm proposed in Balbin and Ramamohanarao [BR].

11. Optimizations and non-redundancy II

As in Section 7 we would like now to consider some specialized versions of the algorithms, this time of Algorithms 7,8,9. It is clear how to extend the definition of strictness to the operators of the form of T_i.

Suppose now that $m = m_1 + m_2$, with $m_1 > 0$ and $m_2 \geq 0$, where for $i = 1, \ldots, m_1$, $k_i > 0$ and T_i is strict, and for $i = m_1, \ldots, m_1 + m_2$, $k_i = 0$. Then each T_i, for $i = m_1 + 1, \ldots, m_1 + m_2$, is a constant.

We can then compute the least fixpoint of the operator T defined by (**) more efficiently by avoiding the repeated generation of $T_{m_1+1}, \ldots, T_{m_1+m_2}$ during the computation of the powers of T, and instead generating them only once, at the beginning of the computation.

We consider here only the appropriate modification of Algorithm 9 leaving similar modifications of Algorithms 7 and 8 to the reader. The following observation will be needed here.

Note 30. Assume the notation of Theorem 24. Then for $j = 2, \ldots, k_i$ the following implication holds:

$$(U(i,j,n))_{j-1} = \emptyset \rightarrow \forall h (j < h \leq k_i \rightarrow (U(i,h,n))_{j-1} = \emptyset)$$

Proof. By definition for $h = j, \ldots, k_i$

$$(U(i,h,n))_{j-1} = (T\uparrow(n-1))_{i,j-1}. \quad \square$$

Let

$$Z = \bigcup_{i=m_1+1}^{m_1+m_2} T_i.$$

The following algorithm is obtained by using Note 30 together with the assumption that each operator T_i, for $i = 1, \ldots, m_1$, is strict

Algorithm 10

```
{$n := 1;$}
$s := \emptyset;$
$t := Z;$
$\Delta t := Z;$
$\Delta\Delta t := Z;$
repeat {$s = T\uparrow(n-1) \wedge t = T\uparrow n \wedge \Delta t = \Delta T\uparrow n$}
    $i := 1;$
    while $i \neq m_1 + 1$ do {$\Delta\Delta t = \bigcup_{l=1}^{i-1} \bigcup_{j=1}^{k_l} T_l(U(l,j,n)) \cup Z$}
    $j := 1;$
    $u[i] := \{t\},[j/\{\Delta t\}_i];$
    if $\forall l(1 \leq l \leq k_i \rightarrow (u[i])_l \neq \emptyset)$ then
    while $j \neq k_i + 1 \wedge (j = 1 \vee (u[i])_{j-1} \neq \emptyset)$ do {$u[i] = U(i,j,n)$};
      if $(u[i])_j \neq \emptyset$ then $\Delta\Delta t =$
      $\Delta\Delta t \cup T(u[i]);$
      $u[i] := u[i][j/\{s\}_i][j+1/\{\Delta t\}_i];$
      $j := j + 1$
    od;
    $i := i + 1;$
  od; {$\Delta\Delta t = \bigcup_{i=1_j}^{m} \bigcup_{j=1}^{k} T(U(i,j,n))$}
```

```
    $s := t$;
    $\Delta t := \Delta\Delta t \setminus t$;
    $t := t \cup \Delta t$;
    $\Delta\Delta t := Z$;
    $\{n := n + 1\}$
**until** $\Delta t = \emptyset$
$\{t = T\uparrow n \wedge \Delta T\uparrow n = \emptyset\}$
$\{t = lfp(T)\}$
```

As in Section 7 we would like now to show that in some sense Algorithm 10 is non-redundant, which should be now interpreted as a statement that it leads to a computation in which each of the operators $T_1, \ldots, T_{m_1}$ is continuously applied to new singleton arguments. Again, to make this idea precise, we assume that each lattice L_i is of the form $L(A_i)$ for some finite set A_i.

Because of the introduced modifications, each operator T_i is applied only to the arguments with all components different from $\emptyset$. Moreover, all these arguments are of the form $U(i,j,n)$, each time for a different pair (j,n). The desired conclusion now follows by Lemma 25 in conjunction with Lemma 19.

To apply these finding to deductive databases it suffices to note the following.

THEOREM 31.
i) *For a unit clause C, T_C is a constant. In fact,*

$$T_C = \{A : A \text{ is a ground instance of } C\}.$$

ii) *For a rule C, T_C is strict.*

PROOF. Straightforward. □

Thus, given a program P (with at least one rule) we can use Algorithm 10 to compute the least fixpoint of T_P by putting

$$Z = \bigcup\{T_C\colon C \text{ is a unit clause}\}$$

and choosing for $T_1, \ldots, T_{m_1}$ the operators T_C with C ranging over all rules from P.

Admittedly, Algorithms 7-10 are not too easy to grasp because of the (difficult to avoid) notational problems. However, when applied to logic programs these algorithms are actually quite easy to understand and analyze. Consider the following, well known program computing in a relation *tr* the transitive closure of a relation *r*.

$$P = A \cup$$
$$\{tr(x,y) \leftarrow r(x,y),$$
$$tr(x,y) \leftarrow tr(x,z),\ tr(z,y)\},$$

where A is a finite, non-empty set of ground unit clauses using the relation r. Then a modification of Algorithm 7 along the lines suggested at the beginning of this section leads after some obvious simplifications to the following algorithm.

Algorithm 11

```
$s := \emptyset$;
$tr := A$;
$\Delta tr := A$;
$\Delta\Delta tr := A$;
**repeat**
```

```
    $\Delta\Delta tr := \Delta\Delta tr \cup \Delta tr(x,z) \bowtie tr(z,y) \cup tr(x,z) \bowtie \Delta tr(z,y)$
    $s := tr$;
    $\Delta tr := \Delta\Delta tr \setminus tr$;
    $tr := tr \cup \Delta tr$;
    $\Delta\Delta tr := A$
**until** $\Delta tr = \emptyset$
```

Here the use of variables together with the join operation $\bowtie$ is used to express the appropriate set operations on relations.

In turn, Algorithm 10 leads to a modification of the above algorithm with the second assignment to $\Delta\Delta tr$ replaced by

$$\Delta\Delta tr := \Delta\Delta tr \cup \Delta tr(x,z) \bowtie tr(z,y) \cup s(x,z) \bowtie \Delta tr(z,y).$$

This modification is non-repetitive, as we have shown on an abstract level.

ACKNOWLEDGEMENTS
This work was partially supported by MCC at Austin, Texas. We would like to thank Carlo Zaniolo for helpful and stimulating discussions on the subject of this paper.

REFERENCES

[BaR] I. BALBIN and K. RAMAMOHANARAO, *A Generalization of the Differential Approach to Recursive Query Evaluation*, Journal of Logic Programming, Vol. 4, pp. 259-262, 1987.

[BR] C. BEERI and R. RAMAKRISHNAN, *The Power of Magic*, in: Proc. 6th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems, pp. 269-283, 1987.

[vEK] M. VAN EMDEN and R. KOWALSKI, *The Semantics of Predicate Logic as a Programming Language*, Journal of ACM, Vol. 23, No. 4, pp. 733-742, 1976.

[OG] S. OWICKI and D. GRIES, *Verifying properties of parallel programs: an axiomatic approach*, Communications of ACM, Vol. 19, No. 5, pp. 279-285, 1976.

[RLK] J. ROHMER, R. LESCOEUR and J.M. KERISIT, *The Alexander Method, a Technique for the Processing of Recursive Axioms in Deductive Databases*, New Generation Computing, Vol. 4, No. 3, pp. 273-285, 1986.

[U] J. ULLMAN, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1988.