The Powerset Algebra as a Natural Tool
to Handle Nested Database Relations

MARC GYSSENS
Department WNI, University of Limburg (LUC), Universitaire Campus,
B-3590 Diepenbeek, Belgium

AND

DIRK VAN GUCHT
Computer Science Department, Indiana University, Bloomington, Indiana 47405-4101

Received October 21, 1988; revised June 27, 1990

The nested relational algebra is often inadequate as a tool to handle nested relations, since
several important queries, such as transitive closure, cannot expressed by it. On the other
hand, the powerset algebra does allow the expression of transitive closure, but in a too
expensive way. Therefore, we consider various other extensions to the nested algebra, such as
least fixpoint and programming constructs, and show the query languages obtained in this
way to be equivalent to (a slight extension of) the powerset algebra, thus both emphasizing
the naturalness and strength of the latter as a tool to manipulate nested relations, and, at the
same time, indicating better ways to implement the powerset algebra.

1. Introduction

In the last several years, much attention has been paid to nested relations. When Codd introduced
the relational database model in 1970–1971 [7, 8], he required relations to be in first normal form, meaning that only “atomic” (i.e., non-
structured) values were allowed in a database. In order to model some database applications more naturally, Makinouchi [18] proposed to generalize the relational
model by removing Codd’s first normal form assumption, thus allowing relations
with set-valued attributes. Subsequently, a generalization of the relational algebra
to relations with set-valued attributes was introduced by Jaeschke and Schek [15].
More specifically, they presented the nest and the unnest operator as tools to
restructure such relations. Finally, Thomas and Fisher [24] generalized this model
by allowing nested relations of arbitrary (but fixed) depth.

Although the nested algebra has been shown in [10, 25] to be complete in the
sense of Bancilhon and Paredaens [3, 4, 20], it suffers from the same disadvantages
as the traditional “flat” relational algebra [21]. For instance, it is impossible to
express transitive closure in the nested algebra [2, 21]. However, the flat algebra is equivalent to several other query languages, such as the tuple and the domain calculus. This property made Codd propose the flat algebra as a standard for expressiveness of flat relational query languages. Calculus-like query languages were also proposed for nested relations [1, 16, 19, 21, 22]. Contrary to the flat case, however, these calculi are either very involved or considerably richer than the nested algebra.

In an algebraic language for database logic, Kuper and Vardi [16] introduced the powerset operator. Recently, much attention has been paid to the powerset algebra obtained by adding this operator to the nested algebra [1, 11-13, 17]. In particular, Abiteboul and Beeri [1] were able to show that the powerset algebra was equivalent to their reasonably "simple" calculus language, and, for that matter, to all "natural" calculi for nested relations.

In this paper, we investigate other extensions of the nested relational algebra. In order to be able to express transitive closure, one can augment the nested algebra with a least fixpoint operator. We show that the least fixpoint closure of the nested algebra is equivalent to the powerset algebra. (A similar result has been shown independently by Abiteboul and Beeri for their calculus [1].) Next, we add commonly used programming constructs to the nested algebra, such as if–then–else tests, while-loops, and for-loops. It turns out that if–then–else tests do not augment the expressive power of the nested algebra; the query languages respectively obtained by adding while-loops and for-loops, however, will be shown as well to be equivalent to the powerset algebra, provided a slight adaptation is made to take into account undefinedness inherent to neverending while-loops.

This result is remarkable for the following reason. In [4], Chandra and Harel introduced QL, a query language which is able to express all computable queries in the context of flat relations. In [5], they discussed RQL, which is a restriction of QL. Both languages contain programming constructs, such as the ones we consider for nested relations. Chandra and Harel proved that RQL is at least as expressive as the least fixpoint closure of flat relational query languages. They furthermore observed that showing that RQL is equivalent in expressive power to the fixpoint closure of a flat relational query language would imply that \( PTIME = PSPACE \), which is generally believed to be false. Hence our results indicate there is a sharp distinction between the properties of query languages for the nested relational model and the standard flat relational model.

All the equivalences established in this article definitely underline both the strength and the naturalness of the powerset algebra as a query language for nested relations. Furthermore, they also indicate ways to implement the powerset algebra.

This paper is organized as follows. In Section 2, we begin by defining a nested algebra similar to the one introduced in [24]. Then, in Section 3, we present the powerset operator and add it to the nested algebra. In Section 4, we consider the least fixpoint operator and establish the equivalence of the least fixpoint closure of the nested algebra, the powerset algebra and the least fixpoint closure of the powerset algebra. In Section 5, we introduce if–then–else tests in the nested algebra.
and show that this extension does not yield increased expressiveness. Next, in Section 6, we slightly extend the nested algebra and the powerset algebra to deal with undefinedness. We then introduce while-loops and show that the while-closures of the nested algebra, the extended nested algebra, the powerset algebra, and the extended powerset algebra are all equivalent to the extended powerset algebra. Finally, in Section 7, we show that considering for-loops in which the loop is bounded by the size of some relation yields similar results as for while-loops.

2. A Model for Working with Nested Relations

2.1. Nested Relations

In the traditional relational model of Codd [7, 8], a database consists of a collection of relations, which are essentially flat tables. Consider the following example.

Example 2.1.1. Below, we show a table that represents a relation from a (simplified) social security database indicating persons with their jobs and the sites where these jobs are executed.

\[
\begin{array}{|c|c|c|}
\hline
\text{PERSON} & \text{JOB} & \text{CITY} \\
\hline
\text{Jeff Willows} & \text{professor} & \text{Austin} \\
\text{Jeff Willows} & \text{president} & \text{Austin} \\
\text{Jeff Willows} & \text{consultant} & \text{Dallas} \\
\hline
\end{array}
\]

Note that in this relation we cannot include persons having no job without using null values [9, 27]. Furthermore, this flat relation does not express the natural "hierarchy" that exists in the various relationships between the attributes.

Both of the drawbacks of the flat relational model identified above can be overcome by representing the data in a nested relation. In such a relation, tuple values need not be atomic, but can in turn be relations.

Example 2.1.2. Consider the following table representing a nested relation:

\[
\begin{array}{|c|c|c|}
\hline
\text{PERSON} & \{ \{ \text{JOB} \} \} & \text{CITY} \\
\hline
\text{Jeff Willows} & \text{professor} & \text{Austin} \\
& \text{president} & \\
& \text{consultant} & \text{Dallas} \\
\text{Mary Higgins} & & \\
\hline
\end{array}
\]
In this nested relation, there are two levels of nesting: jobs are grouped by the city in which they are executed and pairs of sets of jobs and cities are grouped by the person having these jobs. Note that the first tuple of this nested relation contains the same information as the "flat" relation of Example 2.1.1; the second tuple represents a person having no job.

In a nested relation instance, tuple entries can be either atomic or in turn nested relation instances. In the table of Example 2.1.2, each box represents a nested relation instance; the scheme corresponding to each such instance is the set of attributes aligned with that box. Let us now show how a nested relation is formally defined.

Basically we assume that we have an infinitely enumerable set $U$ of atomic attributes and an infinitely enumerable set $V$ of atomic values. In the remainder of this section, we explain how nested attributes and values, nested relation schemes, nested relation instances, and nested relations are constructed from these.

First, we define a nested attribute. Nested attributes can either be atomic or composed. The latter ones are sets of nested attributes (which can be composed in turn); the values associated to them are relation instances over that set of nested attributes, interpreted as a scheme.

**Definition 2.1.1.** The set of all nested attributes $\mathcal{W}$ is the smallest set containing $U$ such that for each finite subset $X$ of $\mathcal{W}$ in which no atomic attribute appears more than once, $X \in \mathcal{W}$. A nested attribute of $U$ is called an atomic attribute; a nested attribute of $\mathcal{W} - U$ is called a composed attribute.

**Example 2.1.3.** If $A, B, C \in U$ then $\{A, \{B, C\}\}$ is a composed nested attribute, but $\{A, \{A, B\}\}$ is not (because $A$ occurs twice).

In the traditional flat relational model, a relation scheme is a set of attributes. In the same philosophy, we define nested relation schemes as follows:

**Definition 2.1.2.** A nested relation scheme $\Omega$ is a composed attribute, i.e., an element of $\mathcal{W} - U$.

Hence a nested relation scheme is a set of nested attributes. If an attribute belonging to a scheme is atomic, then the values associated to that attribute in an instance of that relation are also atomic; if it is composed, then the associated values are in turn nested relation instances whose scheme is precisely the composed attribute under consideration. Let us explain this idea for an example.

**Example 2.1.4.** Reconsider the nested relation in Example 2.1.2. The scheme of that relation is the composed attribute

$$\{\text{PERSON}, \{\{\text{JOB}\}, \text{CITY}\}\}.$$
This scheme contains two attributes: the atomic attribute PERSON and the composed attribute \{\{JOB\}, CITY\}. The nested relation instance represented in Example 2.1.2 consists of two tuples. In these tuples, the values associated to PERSON are atomic; the values associated to \{\{JOB\}, CITY\} are nested relation instances over \{\{JOB\}, CITY\}, which is again a two-attribute scheme. E.g., in the first tuple, the second component is a two-tuple instance; in each of these two tuples, the value corresponding to the composed attribute \{JOB\} is a (flat) relation instance over the one-attribute scheme \{JOB\}, whereas the value corresponding to the atomic attribute CITY is atomic. Also, note that empty instances are allowed as entries of a tuple; e.g., in the second tuple of the relation in Example 2.1.2, Mary Higgins is associated to an empty instance.

We now formally define nested values, nested tuples, and nested relation instances:

**Definition 2.1.3.** The set \(\mathcal{V}\) of all nested values, the set \(\mathcal{I}_X\) of all nested relation instances over \(X \in \mathcal{U} - U\), the set \(\mathcal{F}_X\) of all nested tuples over \(X \in \mathcal{U} - U\), and the set \(\mathcal{I}\) of all nested relation instances are the smallest sets satisfying

1. \(\mathcal{V} = \mathcal{V} \cup \mathcal{I}\);
2. \(\mathcal{I} = \bigcup_{X \in \mathcal{U} - U} \mathcal{I}_X\);
3. \(\mathcal{F}_X\) consists of all finite subsets of \(\mathcal{I}_X\);
4. \(\mathcal{F}_X\) consists of all mappings \(t\) from \(X\) into \(\mathcal{V}\), called nested tuples, satisfying \(t(A) \in \mathcal{V}\) for all atomic attributes \(A \in X \cap U\) and \(t(Y) \in \mathcal{I}_Y\) for all composed attributes \(Y \in X - U\).

We now have all the necessary ingredients to define a nested relation:

**Definition 2.1.4.** A nested relation is a pair \((\Omega, \omega)\), where \(\Omega \in \mathcal{U} - U\) and \(\omega \in \mathcal{I}_\Omega\). \(\Omega\) is called the scheme of the relation and \(\omega\) is called the instance of the relation. If \(\Omega \subseteq U\), then \((\Omega, \omega)\) is called a flat relation.

From now on, we omit the qualification "nested" and refer to the notions defined above as attributes, schemes, values, tuples, instances, and relations for short.

We conclude this subsection with a notational issue. Let \(\Omega\) be a scheme and let \(\mathcal{W}\) be a set of instances over \(\Omega\). Such a set of instances can alternatively be represented as a one-attribute relation, the scheme of which is \(\{\Omega\}\) and the instance of which is \(\{t \in \mathcal{F}_\Omega \mid t(\Omega) \in \mathcal{W}\}\). For notational convenience, we denote this relation as \((\{\Omega\}, \mathcal{W})\).

### 2.2. The Nested Algebra

In this section, we define a nested algebra, similar to the one in [24], based on the model for relations described in the previous section. It is generated by eight operators, defined below. Basically, these operators are borrowed from the classical "flat" relational algebra, except for nesting and unnesting. However, some
technicalities are unavoidable to fit renaming and selection into the context of nested relations.

**Definition 2.2.1.** Let \((\Omega, \omega), (\Omega_1, \omega_1), (\Omega_2, \omega_2)\) be relations. Suppose that the sets of atomic attributes from which \(\Omega_1\) and \(\Omega_2\) are built, are disjoint.

- The **union** \((\Omega, \omega) \cup (\Omega_2, \omega_2)\) equals \((\Omega, \omega_1 \cup \omega_2)\).
- The **difference** \((\Omega, \omega_1) - (\Omega_2, \omega_2)\) equals \((\Omega, \omega_1 - \omega_2)\).
- The **cartesian product** \((\Omega_1, \omega_1) \times (\Omega_2, \omega_2)\) equals \((\Omega', \omega')\), where \(\Omega' = \Omega_1 \cup \Omega_2\) and
  \[
  \omega' = \{ t \in \mathcal{F}_{\Omega'} | t|_{\Omega_1} \in \omega_1 \land t|_{\Omega_2} \in \omega_2 \}.
  \]
- Let \(\Omega' \subseteq \Omega\). The **projection** \(\pi_{\Omega'}(\Omega, \omega)\) equals \((\Omega', \omega')\), where \(\omega' = \{ t|_{\Omega'} | t \in \omega \}\).
- Let \(X \subseteq \Omega\). The **nesting** \(\nu_X(\Omega, \omega)\) equals \((\Omega', \omega')\), where \(\Omega' = (\Omega - X) \cup \{ X \}\) and
  \[
  \omega' = \{ t \in \mathcal{F}_{\Omega'} | \exists t' \in \omega : t|_{\Omega - X} = t'|_{\Omega - X} \land t(X) = \{ t'' | t'' \in \omega \land t'|_{\Omega - X} = t''|_{\Omega - X} \} \}.
  \]
- Let \(X \in \Omega - U\). The **unnesting** \(\mu_X(\Omega, \omega)\) equals \((\Omega', \omega')\), where \(\Omega' = (\Omega - \{ X \}) \cup X\) and
  \[
  \omega = \{ t \in \mathcal{F}_{\Omega'} | \exists t' \in \omega : t|_{\Omega - \{ X \}} = t'|_{\Omega - \{ X \}} \land t|_X = t'(X) \}.
  \]

Let \((\Omega, \omega)\) be a relation scheme. Let \(\varphi\) be a permutation on \(U\). \(\varphi\) is extended in the natural way to \(\mathcal{H}, \mathcal{F}, \mathcal{S},\) and \(\nu:\)

- The **renaming** \(\rho^\varphi(\Omega, \omega)\) equals \((\varphi(\Omega), \varphi(\omega))\);
- Assume furthermore that \(\varphi(\Omega) = \Omega\). The **selection** \(\sigma^\varphi(\Omega, \omega)\) equals \((\Omega, \omega')\), where
  \[
  \omega' = \{ t \in \omega | \varphi(t) = t \}.
  \]

Observe that all operations, including renaming and selection, only work at the highest level of nesting. The non-standard way in which renamings and selections are denoted serves to avoid ambiguities. Indeed, if \(X\) and \(X'\) are composed attributes, a notation such as \(\rho_{X \leftarrow X'}\) for the renaming of \(X\) to \(X'\) does not indicate which atomic attributes of \(X\) should be renamed to which atomic attributes of \(X'\). Explicitly specifying this is actually defining a permutation on \(U\).

However, if in subsequent examples or constructions it is clear as to how the renaming is done, we will nevertheless use the less involved notation \(\rho_{X \leftarrow X'}\). We use the same notation if \(X\) is a set of attributes of the scheme under consideration, and
each attribute of \( X \) is renamed in a well known way to an attribute of \( X' \). Similarly, if selection comes down to only checking whether the values for composed attributes \( X \) and \( X' \) are equal upon renaming and if no ambiguity is possible as to how the atomic attributes in \( X \) and \( X' \) are to be matched, we denote this selection by \( \sigma_{X \leftarrow X'} \). Again, we use the same notation if \( X \) and \( X' \) are sets of attributes of the scheme under consideration.

Finally, note that the cartesian product is only defined for relations with completely "independent" schemes. This is actually not a severe restriction: it is indeed always possible to arrange that the schemes of two relations have no atomic attributes in common by performing an appropriate renaming.

**Example 2.2.1.** Reconsider the relation in Example 2.1.1. If we denote this relation by \((\Omega, \omega)\), then \(\mu_{\{\text{JOB}, \text{CITY}\}}(\Omega, \omega)\) yields a relation with scheme \(\Omega' = \{\text{PERSON}, \{\text{JOB}\}, \text{CITY}\}\) which can be represented as

\[
\begin{array}{ccc}
\text{PERSON} & \{\text{JOB}\} & \text{CITY} \\
\hline
\text{Jeff Willows} & \text{professor} & \text{Austin} \\
\text{Jeff Willows} & \text{consultant} & \text{Dallas}
\end{array}
\]

If we nest this last relation over \(\{\{\text{JOB}\}, \text{CITY}\}\), we again obtain a relation over \(\Omega\) the instance of which corresponds to the first tuple of \(\omega\). Note that in general, an unnesting cannot be undone by a nesting, even if no empty composed values are present. A nesting on the other hand, can always be undone by the corresponding unnesting.

We can now define a *nested algebra expression* (nae):

**Definition 2.2.2.** 1. The variables \(x, y, z, \ldots\) are naes;
2. For all \(\Omega \in \mathcal{U} - \emptyset\), \((\Omega, \emptyset)\) is a nae;
3. For all \(\Omega \in \mathcal{U} \cup \emptyset\), \((\{\Omega\}, \{\emptyset\})\) is a nae;
4. For all naes, the basic operators of Definition 2.2.1 applied to them, are also naes, provided these new expressions make sense.

We implicitly assume that all variables are typed, i.e., associated to relations having one particular scheme. For simplicity's sake, however, we will not reflect this fact in our notation.

Note that the only "constants" we allow in naes are empty relations (and nested variations of them). These very natural constant relations (and those that can be built from them using nest and unnest) suffice for marking purposes, as will be seen in the sequel. As to other constants, we took the standpoint that, in principal, no
value is preferential to another one. However, constant values can be simulated in our model by introducing them in separate, singleton relations.

The expressions introduced in items 1, 2, and 3 of Definition 2.2.2 will be called primitive expressions. The set of all names will be denoted by \( \mathcal{N} \). If \( E(x, y, ...) \in \mathcal{N} \) and \( r, s, ... \) is a finite sequence of relations the schemes of which are “compatible” with the variables of the expression, then \( E(r, s, ...) \) is interpreted as the relation obtained by substituting every occurrence of a variable in \( E(x, y, ...) \) by the corresponding relation. Obviously, the scheme of \( E(r, s, ...) \) does not depend on the actual instances of \( r, s, ... \). Therefore we often denote this scheme as \( \Omega^E \).

To avoid extensive use of brackets in the sequel, we assume the following precedence on nested algebra operators:

1. unary operators;
2. cartesian product;
3. set operators.

3. The Powerset Operator and the Powerset Algebra

Recently, much attention has been paid to the expressiveness of the nested relational algebra and related query languages [1, 12–14, 17, 21]. In order to deal with this problem, it suffices to consider single relations only, since a database can always be represented as the cartesian product of its non-empty members. In its most general form, the question that must be asked is [4, 5]: Let \( Q \) be a computable query, i.e., a partial recursive mapping from relations to relations that preserves isomorphism. Does there exist \( E(x) \in \mathcal{N} \) such that \( \forall r \colon E(r) = Q(r) \)? Although it has been shown [10, 25] that it is always possible to find an expression that satisfies this equality for any particular relation, there is in general no expression that will do for all relations. E.g., the transitive closure of a binary flat relation, which is not expressible in the classical relational algebra, is also not expressible in the nested algebra [21].

Therefore, several attempts have been made to enrich the nested relational algebra. One of these consists of adding the powerset operator to the nested algebra. This operator was introduced by Kuper and Vardi in [16] as one of the primitive operators in their algebraic query language for database logic. Basically the powerset operator generates all subsets of a given relation:

**Definition 3.1.** Let \( (Q, \omega) \) be a relation. Let \( 2^\omega \) denote the set of all subsets of \( \omega \). Then (using the notation introduced at the end of Section 2) the powerset \( \Pi(Q, \omega) \) equals \( (\{Q\}, 2^\omega) \).

**Example 3.1.** Reconsider the relation \( (\Omega, \omega) \) represented in Example 2.1.2. \( \Pi(Q, \omega) \) is a one-attribute four-tuple relation with scheme \( \{\{PERSON\}, \{\{JOB\}, CITY\}\} \) which can be represented as
Using a combinatorial argument, it was established in [11]:

**Theorem 3.1.** There is no nqe that expresses the powerset operator.

We now consider the **powerset algebra** whose set of expressions $\mathcal{P}$ is generated by the basic operators of the nested algebra defined in Definition 2.2.1, augmented with the powerset operator. An expression of the powerset algebra is called a **powerset algebra expression** (pae). Although only one operator is added, $\mathcal{P}$ turns out to be remarkably more expressive than $\mathcal{N}$ [1, 11].

For instance, it is possible to express transitive closure in the powerset algebra. Indeed, in [26] it is shown that transitive closure is in the existential-second-order calculus for relations, while in [16] it is shown that the nested calculus (which subsumes existential-second-order) is equivalent to the LDM algebra, which is is essentially equivalent to the powerset algebra for nested relations. It is also possible to show this directly. Transitive closure is a particular case of a least fixpoint query, and, in Lemma 4.6, we show how least fixpoint queries can be expressed in the powerset algebra. For such queries, the powerset operator allows one to compute all "possible" answers; using the other operators, it is then possible to "select" the correct answer.

Since actually using the powerset operator is very expensive, we examine other possible ways to enrich the nested algebra. We show that the query languages thus obtained are equivalent to (a slight extension of) the powerset algebra w.r.t.
expressive power, thus establishing the naturalness of the powerset algebra as a standard for the expressive power of a query language on nested relations, and, at the same time, indicating better ways to implement such a query language.

4. AUGMENTING THE ALGEBRA WITH A LEAST FIXPOINT OPERATOR

Another powerful tool to model relational queries is the least fixpoint operator [2, 5, 6]. Actually, this operator does not work on relations but on queries; it transforms them into other ones. The initial motivation for introducing a least fixpoint construct in the traditional flat algebra was the observation by Aho and Ullman [2] that the transitive closure could not be expressed in the algebra. Augmenting the algebra with a least fixpoint operator then seems the most obvious extension to the algebra that can handle transitive closure. Since the nested algebra cannot express transitive closure [21], it is straightforward to consider the closure of the nested algebra under least fixpoint as well. In this section, we show that the query language thus obtained is equally as expressive as the powerset algebra. Independently, Abiteboul and Beeri [1] obtained similar results for their calculus.

We first point out for which queries in the nested algebra we will consider the least fixpoint operator:

**DEFINITION 4.1.** An \( \text{lfp expression} \) is a unary scheme preserving expression \( E(x) \) such that for all relations \( r \) and \( s \) for which \( E(r) \) and \( E(s) \) are defined,

1. \( r \subseteq E(r) \) (increasing); \(^1\)
2. \( r \subseteq s \) implies \( E(r) \subseteq E(s) \) (monotone).

For each lfp expression \( E(x) \) we consider another expression we denote as \( E^*(x) \). If \( r \) is a relation, then \( E^*(r) \) is defined if and only if \( E(r) \) is defined and must in that case be interpreted [23] as the smallest relation \( s \) containing \( r \) for which \( E(s) = s \). A straightforward inductive argument shows that \( E^*(r) = \bigcup_{i=1}^{\infty} E(i)(r) \), where, for all positive \( i \), \( E^i(x) \) stands for

\[
\underbrace{E \cdots E}_{i \text{ times}}(x)
\]

Note that for each relation \( r \) for which \( E(r) \) is defined, \( E^*(r) \) can always be computed, since for some positive integer \( k \), \( E^k(r) = E^{k+1}(r) = E^{k+2}(r) = \cdots \).

We now formally define the lfp closure of the nested algebra (respectively the powerset algebra):

**DEFINITION 4.2.** The lfp closure of the nested algebra \( \mathcal{N} \) (respectively the powerset algebra \( \mathcal{P} \)) is the smallest set \( \mathcal{N}^* \) (respectively \( \mathcal{P}^* \)) satisfying

1. \( \mathcal{N}^* \) (respectively \( \mathcal{P}^* \)) contains \( \mathcal{N} \) (respectively \( \mathcal{P} \));

\(^1\) For \( r_1 = (\Omega, \omega_1) \) and \( r_2 = (\Omega, \omega_2) \), we write \( r_1 \subseteq r_2 \) for \( \omega_1 \subseteq \omega_2 \).
2. For each lfp expression $E(x)$ in $N^*$ (respectively $P^*$), $E^*(x)$ is also in $N^*$ (respectively $P^*$);

3. Whenever $E_1(x)$ and $E_2(y)$ are in $N^*$ (respectively $P^*$), then $E_2 E_1(x)$ is also in $N^*$ (respectively $P^*$), provided this expression makes sense. In other words, $N^*$ (respectively $P^*$) is closed under substitution.

An expression of $N^*$ (respectively $P^*$) is called an lnae (respectively a lpae).

A classical example of a query that can be constructed from an algebra query using the lfp operator is, as mentioned, the transitive closure of a flat binary relation:

**Example 4.1.** Let $r - (\{A, B\}, \omega)$ be a flat relation. Consider the following expression:

$$E(x) = x \cup \pi_{\{A, B\}} \sigma_{C = D} (\mu_{C - B}(x) \times \nu_{D - A}(x)).$$

$E(x)$ is unary, scheme preserving, increasing, and monotone, and, therefore, an lfp expression. Obviously the transitive closure of $r$ equals $E^*(r)$.

As this query cannot be expressed in the nested algebra, it follows that the lfp closure of the nested algebra is strictly more expressive than the ordinary nested algebra. Actually, the lfp closure of the nested algebra is equivalent to the powerset algebra (and hence also to the lfp closure of the powerset algebra). We first show that the powerset operator can be expressed in the lfp closure of the nested algebra:

**Lemma 4.1.** There exists an lnae that expresses the powerset operator.

**Proof.** Let $r$ be the relation $(\Omega, \omega)$. Let $\varphi$ and $\psi$ be permutations on $U$ such that $\Omega, \Omega^{\varphi} - \varphi(\Omega)$ and $\Omega^{\psi} - \psi(\Omega)$ have no atomic attributes in common. Since $\Pi(r)$ is a relation over $\{\Omega\}$, we first need an nae $E_1(x)$ such that $E_1(r)$ is a relation with scheme $\{\Omega\}$:

$$E_1(x) = \pi_{\{\Omega\}} v_{\Omega} \sigma_{\Omega = \Omega^{\varphi}} (x \times \mu_{\Omega^{\varphi} - \Omega}(x)) \cup (\{\Omega\}, \{\emptyset\}).$$

$E_1(r)$ consists of all singletons of $r$ and the empty set, i.e., of all subsets of $r$ of size at most 1. We now write down an expression $E_2(x)$, defined on relations with scheme $\{\Omega\}$:

$$E_2(x) = \pi_{\{\Omega\}} v_{\Omega} \mu_{\Omega} (\sigma_{\Omega = \Omega^{\varphi}} (x \times \mu_{\Omega^{\varphi} - \Omega}(x) \times \mu_{\Omega^{\psi} - \Omega}(x))) \cup (\{\Omega\}, \{\emptyset\}).$$

If $s$ consists of all subsets of $r$ up to size $i$, then $E_2(s)$ consists of all subsets of $r$ up to size $2i$. Since $E_2(x)$ is an lfp expression, we may conclude

$$\Pi(x) = E_2^* E_1(x).$$
We now show that for each lpae there exists an equivalent pae. Therefore we need four technical lemmas (Lemmas 4.2 to 4.5).

**Definition 4.3.** Let \( \{\Omega\}, \omega \) be a relation. Then the global unnesting \( M(\{\Omega\}, \omega) \) equals \( \mu_{\Omega}(\{\Omega\}, \omega) \).

Let \( \Omega, \omega \) be a relation. Then the global nesting \( N(\Omega, \omega) \) equals \( (\{\Omega\}, \{\omega\}) \).

**Lemma 4.2.** There exist naes that express the global unnesting and the global nesting.

*Proof.* For the global unnesting, Lemma 4.2 is a trivial consequence of Definition 4.3. Let us therefore concentrate on the global nesting. Let \( \Omega, \omega \) be a relation. Let \( \phi \) be a permutation on \( U \) such that \( \Omega \) and \( \Omega^\phi = \phi(\Omega) \) have no atomic attributes in common (\( \phi \) is extended to \( \Omega \), \( \mathcal{F} \), and \( \mathcal{V} \) in the usual way). By considering the cases \( \omega = \emptyset \) and \( \omega \neq \emptyset \) it is readily verified that

\[
N(x) = \mu(\Omega) \pi_{\{\Omega\}} \sigma_{(\Omega^\phi)} = (\Omega)(v|_{\Omega})(v_{\Omega}(x) \cup (\{\Omega\}, \{\emptyset\}))
\]

always returns the correct result. 

**Definition 4.4.** Let \( r = (\Omega, \omega) \) be a relation and let \( X, X' \in \Omega - U \) be composed attributes. Suppose there exists a permutation \( \phi \) on \( U \) such that \( X' = \phi(X) \). \( \phi \) is extended to \( \mathcal{F} \) in the usual way. Then the inclusion selection \( \sigma_{X' \subseteq X}(r) \) (or \( \sigma_{X \subseteq X'}(r) \) if \( \phi \) is understood) equals the relation

\[
(\Omega, \{t \in \omega | \phi(t(X)) \subseteq t(X')\}).
\]

**Lemma 4.3.** Let \( r, X, X', \phi \) be as in Definition 4.4. There exists an nae that expresses \( \sigma_{X' \subseteq X}(r) \), independent of \( \omega \).

*Proof.* Let \( \psi \) be a permutation on \( U \) such that \( \Omega \) and \( \Omega^\psi = \psi(\Omega) \) have no atomic attributes in common. Let \( X^\psi = \psi(X) \) and let \( X'_\psi = \psi(X') \). We invite the reader to check that the following nae satisfies the requirements of Lemma 4.3:

\[
\sigma_{X' \subseteq X}(r) = \mu_{\Omega} \sigma_{X' = X} v_{X'} \pi_{\Omega \cup X^\psi} \sigma_{X^\psi = X'} \psi \mu_{X'}
\]

\[
\mu_{X^\psi} \sigma_{X^\psi = X^\psi} (x \times \rho_{\Omega^\psi \rightarrow \Omega}(x)) \cup \pi_{\Omega} \sigma_{X = X'} (x \times (\{X^\psi\}, \{\emptyset\})).
\]

**Definition 4.5.** Let \( E(x) \) be a pae defined on relations \( r \) with scheme \( \Omega \). Let \( \Omega^E \) be the scheme of the resulting relations \( E(r) \). Let \( \phi \) be a one to one mapping from \( U \) to itself such that no atomic attribute of \( \Omega \) is contained in the range of \( \phi \) and let \( \Omega^\phi = \phi(\Omega) \). Let \( s = (\{\Omega\}, \omega) \) be a relation with scheme \( \{\Omega\} \). Then we define

\[
\bar{E}(s) = (\{\Omega, \Omega^E\}, \{t \in \mathcal{F}_{\{\Omega, \Omega^E\}} | t(\Omega) \in \omega \& (\Omega^E, t(\Omega)) = \phi(E(\Omega, t(\Omega)))\}).
\]
Intuitively, $\bar{E}$ applies $E$ to each relation contained in $s$. We have:

**Lemma 4.4.** Using the notations in Definition 4.5, there exists a pae that expresses $\bar{E}(s)$, independent of $\omega$.

**Proof.** By induction on the size of $E(x)$. As an example, let $E(x) = E_1(x) \times E_2(x)$. Assume that $\bar{E}_1(y)$ and $\bar{E}_2(y)$ are paes satisfying Lemma 4.4 for $E_1$ and $E_2$, respectively. Let $\psi$ be a permutation on $U$ such that $\Omega' = \psi(\Omega)$, $\Omega_{\psi \circ \omega}^{E_1} = \psi(\Omega_{\omega}^{E_1})$, and $\Omega_{\psi \circ \omega}^{E_2} = \psi(\Omega_{\omega}^{E_2})$ have no atomic attributes in common with $\Omega$, $\Omega_{\omega_1}^{E_1}$, and $\Omega_{\omega_2}^{E_2}$. Then

$$
\bar{E}(y) = \pi_{\Omega', \Omega_{\omega_1}^{E_1} \cup \Omega_{\omega_2}^{E_2}} \left( \pi_{\Omega_{\omega_1}^{E_1}} \left( \bar{E}_1(y) \times \rho_{\omega, \omega_1} \bar{E}_2(y) \right) \right)
$$

satisfies Lemma 4.4 for the expression $E$.

We leave it to the reader to write similar expressions for the other nested algebra operators as well as for paes of size 1, which are all fairly straightforward.

Finally, let $E(x) = \Pi E_1(x)$, and let $\bar{E}_1(y)$ be the pae satisfying Lemma 4.4 for $E_1$. Let $\psi$ be a permutation on $U$ satisfying the same conditions for $S_2$ and $S_3$ as above. Then

$$
\bar{E}(y) = \pi_{\Omega', \Omega_{\omega_1}^{E_1}} \left( \pi_{\Omega_{\omega_1}^{E_1}} \left( \bar{E}_1(y) \times \rho_{\omega, \omega_1} \bar{E}_2(y) \right) \right)
$$

satisfies Lemma 4.4 for the expression $E$.

**Lemma 4.5.** Let $r$ be as in Definition 4.6. There exists a pae that expresses $E_{\Omega}(r)$, independent of $\omega$.

**Proof.** Let $E_{\omega}(x)$ denote the expression we are going to construct. Rather than writing down this expression which is very involved, we explain how it is constructed. First, we construct an expression which yields a one attribute relation in which all the atomic values of $r$ appear. This expression is obtained by a sequence of unnestings, followed by projection, renaming, and union. Let $E_1(x)$ be this expression. Now, consider for each atomic attribute $A$ in $\Omega$ the expression $E_A(x)$ yielding a relation over $\{A\}$ obtained from $E_1(x)$ by an appropriate renaming. We now construct $E_{\omega}(x)$ inductively as follows. Let $X = \{X_1, \ldots, X_k\}$ be a set of attributes. Then $E_X(x) = \Pi (E_{X_i}(x) \times \cdots \times E_{X_i}(x))$. Obviously, $E_{\omega}(x)$ satisfies the requirements of Lemma 4.5.

We are now ready to show:

**Lemma 4.6.** Each lpaе can be expressed by a pae.
Proof. Let $E(x)$ be a monotone pae. We show that there exists a pae that
expresses $E^*(x)$. Therefore, let $r = (\Omega, \omega)$ be a relation on which $E(x)$ is defined.
Let $\varphi$ be as in Lemma 4.4 and let $\psi_1$ and $\psi_2$ be permutations on $U$ such that $\Omega$, $\Omega_1 = \psi_1(\Omega)$, and $\Omega_2 = \psi_2(\Omega)$ have no atomic attributes in common. We now introduce some paes the last of which expresses $E^*(x)$:

$$E_1(x) = \pi_{\{\Omega_1\}} \sigma_{\Omega - \Omega_1} \mu E_{\Omega}(x).$$

Clearly, $E_1(r)$ consists of the set of all relations $s$ over $\Omega$ satisfying
1. the atomic values of $s$ occur in $r$;
2. $E(s) = s$.

Clearly, of all these relations, $E^*(r)$ is the smallest containing $r$. Therefore, let

$$E_2(x) = \pi_{\{\Omega_1\}} \sigma_{\Omega_1} \rho_{\Omega_1} \mu E_{\Omega}(x).$$

$E_2(r)$ consists of all the relations of $E_1(r)$ that contain $r$. Since the smallest of these
is characterized by its containment in all the relations of $E_2(r)$, we finally have

$$E^*(x) = \mu \pi_{\{\Omega_1\}} \sigma_{\Omega_1} \rho_{\Omega_1} (E_2(x) \times E_1(x)).$$

As an immediate corollary to Lemmas 4.1 and 4.6, we have:

**Theorem 4.1.** The lfp closure of the nested algebra, the powerset algebra, and the
lfp closure of the powerset algebra are equivalent, i.e., they all express the same class
of queries.

In the following two sections, we propose some other possible extensions of the
nested algebra by introducing commonly used programming constructs. We show that

1. If–then–else tests do not increase the expressiveness of the nested algebra
(respectively the powerset algebra);
2. Augmenting the nested algebra with while-loops, respectively for-loops,
yields a query language equivalent to (a slight extension of) the powerset algebra;
3. (A slight extension of) the powerset algebra is closed under while-loops
and for-loops.

5. INTRODUCING IF–THEN–ELSE TESTS IN THE ALGEBRA

First, we formally define an if–then–else query:

**Definition 5.1.** Let $r = (\Omega, \omega)$ be a relation. Let $E_1(x)$, $E_2(x)$, and $E_3(x)$ be
unary expressions, defined on relations with scheme $\Omega$, such that $\Omega^{E_3} = \Omega^{E_1}$. An
if-then-else query of type $\Omega$ is a unary query $Q$ for which $Q(r)$ can be written as the result of a program of the following form:

```
begin
  if $E_1(r) \neq (\Omega^{E_1}, \emptyset)$
  then
    $Q(r) := E_2(r)$
  else
    $Q(r) := E_3(r)$
end.
```

If the expressions in Definition 5.1 are all naes (respectively paes), we say that $Q$ is an if-then-else query in the nested algebra (respectively the powerset algebra). We can show:

**Theorem 5.1.** If-then-else queries in the nested algebra (respectively the powerset algebra) can be expressed in the nested algebra (respectively the powerset algebra).

**Proof.** Consider the if-then-else query of Definition 5.1. Without loss of generality, we may assume that $LP$ and $GE_2 = LP^\perp$ have no atomic attributes in common; otherwise, the appropriate renamings have to be performed. Consider the following expression:

$$E_4(x) = \pi_{\Omega^{E_2}}(E_1(x) \times E_2(x)) \cup (E_3(x) - \pi_{\Omega^{E_3}}(E_1(x) \times E_3(x))).$$

A straightforward verification shows that $Q(r) = E_4(r)$.

Hence the introduction of if-then-else queries does not increase the expressive power of the algebra.

### 6. Introducing While-Loops in the Algebra

In [4], Chandra and Harel introduced the language $QL$ to express queries on flat relations. $QL$ basically consists of the classical flat relational algebra, augmented with two very powerful features: unranked variables (i.e., not associated with a fixed scheme) and a while-construct. Both features give to $QL$ the computing power of Turing machines and hence all computable queries on flat relations can be expressed in $QL$.

Of course, we should not hope that, e.g., the powerset algebra would have the same expressive power as $QL$, since powerset algebra expressions have a fixed scheme. In [5], however, Chandra and Harel introduced and studied the language $RQL$ which is syntactically almost identical to $QL$, but with the restriction that all variables are ranked (i.e., associated to a fixed scheme). Therefore, we wanted to see what happens to the expressive power of the nested algebra when it is augmented with a while-construct. We show in this section that this augmentation results in
exactly the expressiveness of the powerset algebra, provided slight extensions are made to deal with undefinedness, inherent to neverending while-loops.

6.1. The \( v \)-Operator

In Definitions 2.2.2 and 4.2, we defined expressions in such a way that their result for some finite sequence of relations could never be undefined; "illegal" expressions or "illegal" substitutions were simply not allowed. The situation for while-loops, however, is clearly different; although not yet formally introduced, one can readily see that the result of a while-loop can be undefined in case it runs indefinitely. Moreover, the result of a while-loop being undefined can really depend on the actual instances used to start up the while-loop. Therefore, we extend the nested algebra (respectively the powerset algebra) slightly to deal with this situation.

First, for each composed attribute \( X \in \mathcal{U} - U \), we add to \( \mathcal{I} \), i.e., to the set of all relation instances over \( X \), the new value \( ?_X \), which should be interpreted as the "undefined instance" over the scheme \( X \). Note that, by this convention, the values \( ?_X \) cannot appear on a lower level in a nested relation. Of course, we now must point out how the basic operators of the nested algebra (respectively, the powerset algebra) work on these new values. Rather straightforwardly, we agree that whenever a basic operator works on a relation (having the appropriate scheme) with an "undefined instance," then the result is also a relation with an "undefined instance," the scheme of which is defined by the operation.

**Example 6.1.1.** Let \((\Omega, \omega)\) be a relation. Let \(\Omega' \in \mathcal{U} - U\) have no atomic attributes in common with \(\Omega\). Assume furthermore that \(X \subseteq \Omega\). Then

- \((\Omega, ?_\Omega) \cup (\Omega', \omega) = (\Omega, ?_\Omega)\);
- \((\Omega, \omega) \times (\Omega', ?_\Omega) = (\Omega \cup \Omega', ?_{\Omega \cup \Omega'})\);
- \(v_X(\Omega, ?_\Omega) = ((\Omega - X) \cup \{X\}, ?_\Omega - X \cup \{X\})\).

Derived operators can also be extended to "undefined instances." Consider, e.g., the global nesting of Definition 4.3. It is straightforward to define

\[ N(\Omega, ?_\Omega) = (\{\Omega\}, ?_{\{\Omega\}}). \]

Furthermore, this extension to Definition 4.3 is compatible with the expression given for global unnesting in the proof of Lemma 4.2.

Obviously, we need an operator that can generate "undefined instances" in an input-dependent way:

**Definition 6.1.1.** Let \((\Omega, \omega)\) be a relation. The \( v \)-operator is defined by

\[
v(\Omega, \omega) = \begin{cases} (\Omega, \omega) & \text{if } \omega \neq \emptyset \\ (\Omega, ?_\Omega) & \text{if } \omega = \emptyset \\ \end{cases}
\]

\[
v(\Omega, ?_\Omega) = (\Omega, ?_\Omega).
\]
In words, the v-operator (where "v" stands for "undefined") maps each non-empty relation to itself and the empty relation to that newly introduced value which corresponds to the scheme of the relation.

We now consider the extended nested algebra (respectively the extended powerset algebra) whose set of expressions is defined by the basic operators of the nested algebra (respectively the powerset algebra), augmented with the v-operator. These generalized expressions are called extended naes (enaes) (respectively extended paes (epaes)).

However, expressions in which the v-operator occurs are in general rather difficult to handle. Fortunately, it turns out that we only have to consider a subclass of all enaes (respectively epaes) to generate the extended nested algebra (respectively the extended powerset algebra):

**Definition 6.1.2.** A normalized enae (respectively epae) is an enae (respectively epae) \( MvE(x, y, \ldots) \), where \( E(x, y, \ldots) \) is a nae (respectively pae) and "\( M \)" stands for global unnesting (Definition 4.3).

Indeed, since the global nest operator (Definition 4.3) never returns an empty relation, even if the relation on which it is applied is empty, it can be used to "shield" relations from unwanted side-effects of the v-operator. In this way, it is possible to pull back an occurrence of the v-operator to the end of the expression it appears in. Finally, global unnesting can then be used to undo the global nesting. We now formalize this argument, thus justifying the term "normalized."

**Lemma 6.1.1.** The set of all normalized enaes (respectively epaes) has the same expressive power as the extended nested algebra (respectively the extended powerset algebra).

**Proof.** We use induction.

First consider the primitive expressions. If \( x \) is a variable, then \( MvN(x) \) is an equivalent normalized enae. The same statement holds when \( x \) is replaced by either \((\Omega, \emptyset)\) or \((\{\Omega\}, \{\emptyset\})\) for some composed attribute \( \Omega \).

Let us now assume that for some nonnegative integer \( k \), all enaes (respectively epaes) containing up to \( k \) basic operations can be normalized. We now show that all enaes (respectively epaes) containing up to \( k + 1 \) basic operations can be normalized.

- First, consider the expression \( MvE_1(x, y, \ldots) \cup MvE_2(x', y', \ldots) \), where \( E_1(x, y, \ldots) \) and \( E_2(x', y', \ldots) \) are naes (respectively paes). (So \( \Omega_{E_1} = \Omega_{E_2} \).) Now let \( \phi \) and \( \psi \) be permutations on \( U \) such that \( \Omega_{E_1} = \Omega_{E_2}, \quad \Omega_{E_1} = \phi(\Omega_{E_1}), \) and \( \Omega_{E_2} = \psi(\Omega_{E_2}) \) have no atomic attributes in common. Clearly,

\[
Mv(\pi_{\Omega_{E_1}}(N(ME_1(x, y, \ldots) \cup ME_2(x', y', \ldots)) \times \rho_{\Omega_{E_1} - \Omega_{E_1}}E_1(x, y, \ldots)
\times \rho_{\Omega_{E_2} - \Omega_{E_2}}E_2(x', y', \ldots)))
\]
is an equivalent normalized nae (respectively pae). Obviously, the same technique works for difference and cartesian product.

- Now consider the expression $E_2 \cdot \mu \cdot E_1(x, y, \ldots)$, where $E_1(x, y, \ldots)$ is a nae (respectively pae) and $E_2(z)$ is a unary nae (respectively pae). Let $\varphi$ be a permutation on $U$ such that $\Omega^{E_1}$ and $\Omega^{E_2}_{\varphi} = \varphi(\Omega^{E_1})$ have no atomic attributes in common. Then

$$Mv(\pi_{\Omega^{E_1}}(NE_2 \cdot ME_1(x, y, \ldots) \times \rho_{\Omega^{E_1}_{\varphi} \rightarrow \Omega^{E_1}} E_1(x, y, \ldots)))$$

is an equivalent normalized nae (respectively pae). This case covers all unary operators, except the $\lor$-operator.

- Finally, consider the expression $\lor \cdot \mu \cdot E_1(x, y, \ldots)$, where $E_1(x, y, \ldots)$ is a nae (respectively pae). Let $\varphi$ and $\psi$ be permutations on $U$ such that $S_1 = \varphi^\prime$, $S_2 = \psi^\prime$, and $\varphi^\prime \cdot S_1 = \psi^\prime$ have no atomic attributes in common. Then

$$Mv(\pi_{\Omega^{E_1}}(NME_1(x, y, \ldots) \times \rho_{\Omega^{E_1}_{\varphi} \rightarrow \Omega^{E_1}} E_1(x, y, \ldots) \times M \rho_{\Omega^{E_1}_{\psi} \rightarrow \Omega^{E_1}} E_1(x, y, \ldots)))$$

is the desired normalized enae (respectively epae).

6.2. While-Loop Queries

In this subsection, we introduce while-loops in the algebra. Therefore we first define in a formal way what we understand by a while-loop query:

**Definition 6.2.1.** Let $r = (\Omega, \omega)$ be a relation. Let $E_1(x)$ and $E_2(x)$ be unary expressions, defined on relations with scheme $\Omega$, such that $E_2(x)$ is scheme preserving. A **while-loop query** of type $\Omega$ is a unary query $Q$ for which $Q(r)$ can be written as the result of a program of the following form:

```
begin
  s := r;
  while $E_1(s) \neq (\Omega^{E_1}, \emptyset)$
do
    s := E_2(s)
od;
Q(r) := s
end.
```

We assume that $Q(r)$ takes the value $(\Omega, ?_\Omega)$ if the while-loop runs indefinitely.

In Definition 6.2.1, we chose for the while-loop a test of the form $E(s) \neq (\Omega^{E_1}, \emptyset)$. One might also have chose $E(s) \neq E'(s)$, $E(s) = E'(s)$, $E(s) = (\Omega^{E_1}, \emptyset)$, $E(s) = (\Omega^{E_1}, \emptyset)$, $E(s) \neq (\Omega^{E_1}, \emptyset)$, $\ldots$. We leave it to the reader to convince himself that all these choices are equivalent with respect to expressive power.

Note that since, apart from possibly "undefined instances," no new values are introduced during the execution of the while-loop above, it is decidable whether or not this while-loop will end.
If the expressions in Definition 6.2.1 are all naes (respectively enaes, paes, or epaes), we say that \( Q \) is a *while-loop query in the nested algebra* (respectively *the extended nested algebra*, *the powerset algebra*, or *the extended powerset algebra*). By starting from naes (respectively enaes, paes, or epaes) and by recursively associating a unary expression to each while-loop query, one can define the *while-closure of the nested algebra* (respectively *the extended nested algebra*, *the powerset algebra*, or *the extended powerset algebra*).

Obviously, the while-closure of the nested algebra is more powerful than even the extended nested algebra itself:

**Example 6.2.1.** We show that the transitive closure of a binary flat relation can be expressed in the while-loop closure of the nested algebra. Let \( E(x) \) be the expression defined in Example 4.1. Now consider the following while-loop query of type \( \{A, B\} \):

```plaintext
begin
s := r;
while E(s) \neq (\{A, B\}, \emptyset)
do
s := E(s)
end;
\( Q(r) := s \)
```

Then the transitive closure of a relation \( r = (\{A, B\}, \omega) \) equals \( Q(r) \).

As explained earlier, we need enaes and epaes to deal with the occurrence of undefinedness, inherent in the introduction of while-loops. Since we want to show closure-properties, we therefore need to consider while-loop queries in the extended nested algebra respectively the extended powerset algebra. On the other hand, nobody really wants to write while-loop queries in which the \( v \)-operator effectively appears. It is therefore very fortunate that the following property holds:

**Theorem 6.2.1.** The while-closure of the extended nested algebra (respectively the extended powerset algebra) is equivalent to the while-closure of the nested algebra (respectively the powerset algebra).

**Proof.** Consider the program in Definition 6.2.1 for the while-loop query \( Q \) of type \( \Omega \), and suppose that \( E_1(x) \) and \( E_2(x) \) are both naes (respectively epaes). We may assume that \( \Omega \) and \( \Omega^{E_1} \) have no atomic attributes in common. Without loss of generality (Lemma 6.1.1), we may also assume that

\[
E_1(x) = MvE'_1(x) \\
E_2(x) = MvE'_2(x),
\]

where \( E'_1(x) \) and \( E'_2(x) \) are both naes (respectively paes). Clearly, if \( s \) is a relation with scheme \( \Omega \), then \( E_1(s) \) or \( E_2(s) \) take undefined values if and only if \( E'_1(s) \) respectively \( E'_2(s) \) are empty.
Now let $p$ be a relation with scheme $\{\Omega\}$ and consider for $i=1,2$ the following if–then–else queries of type $\{\Omega\}$ in the nested algebra (respectively the powerset algebra):

\[
\begin{align*}
\text{begin} \\
\text{if } p \times E_i M(p) \neq (\{\Omega, \Omega E_i\}, \emptyset) \\
\text{then} \\
Q_i(p) := E_i M(p) \\
\text{else} \\
Q_i(p) := (\{\Omega E_i\}, \emptyset) \\
\text{end.}
\end{align*}
\]

For $i=1,2$, let $E^n_i(y)$ be a nae (respectively pae) that expresses $Q_i$ (Theorem 5.1) and consider the following while-loop query $Q'$ of type $\{\Omega\}$ (taking into account the remarks made about the test condition after Definition 6.2.1):

\[
\begin{align*}
\text{begin} \\
q := p; \\
\text{while } E^n_i(q) \neq (\{\Omega E_i\}, \{\emptyset\}) \\
\text{do} \\
q := E^n_i(q) \\
\text{od;} \\
Q'(p) := q \\
\text{end.}
\end{align*}
\]

$Q'$ obviously is a while-loop query in the nested algebra (respectively the powerset algebra). Now observe that as soon as $q$ becomes empty (i.e., $q = (\{\Omega\}, \emptyset)$), it remains empty, and hence the while-loop runs indefinitely. Furthermore, $q$ becomes empty if and only if $E_1 M(q)$ or $E_2 M(q)$ becomes empty, i.e., if and only if $E_1 M(q)$ or $E_2 M(q)$ takes an undefined value. Hence we may conclude that $Q(r) = MQ'(N(r))$, whence the theorem.

6.3. The Expressive Power of the While-Construct

In this subsection, we show the equivalence of

1. the while-closure of the nested algebra;
2. the while-closure of the extended nested algebra;
3. the while-closure of the powerset algebra;
4. the while-closure of the extended powerset algebra;
5. the extended powerset algebra.

By Theorem 6.2.1, this statement will have been proved if we can demonstrate that

1. The powerset operator can be expressed in the while-closure of the nested algebra;
2. Each while-loop query in the powerset algebra can be expressed in the extended powerset algebra.

The first item is the easier to show:

**Lemma 6.3.1.** The powerset operator can be expressed in the while-closure of the nested algebra.

**Proof.** Let \( r \) be the relation \((\Omega, \omega)\). Let \( \varphi \) and \( \psi \) be permutations on \( U \) such that \( \Omega, \Omega^\varphi = \varphi(\Omega) \), and \( \Omega^\psi = \psi(\Omega) \) have no atomic attributes in common. Since \( \Pi(r) \) is a relation over \( \{\Omega\} \), we first need a nae \( E_1(x) \) such that \( E_1(r) \) is a relation with scheme \( \{\Omega\} \):

\[
E_1(x) = \pi_{\{\Omega\}} \nu_{\sigma_{\Omega^\varphi \cdot \Omega}\sigma_{\Omega^\psi \cdot \Omega}}(x \times \rho_{\Omega^\varphi \cdot \Omega}(x)) \cup \{\Omega\}, \{\varnothing\}\).
\]

If \( \omega \neq ?_\Omega \), then clearly \( E_1(r) \) consists of all singleton subrelations of \( r \) and the empty relation, i.e., of all subrelations of \( r \) of size at most 1.

We now write down an expression \( E_2(y) \), defined on relations with scheme \( \{\Omega\} \):

\[
E_2(y) = \pi_{\{\Omega\}} \nu_{\sigma_{\Omega^\varphi \cdot \Omega}v_{\Omega^\varphi \cdot \Omega}(y \times \rho_{\Omega^\varphi \cdot \Omega}(y))} \cup \sigma_{\Omega^\varphi \cdot \Omega}(y \times \rho_{\Omega^\varphi \cdot \Omega}(y)) \cup \{\Omega\}, \{\varnothing\}\).
\]

Let \( p \) be a relation over the scheme \( \{\Omega\} \). If \( p \) consists of all subrelations of \( r \) up to size \( i \), then \( E_2(p) \) consists of all subsets of \( r \) up to size \( 2i \). Now consider the following while-loop query \( Q \) of type \( \{\Omega\} \):

begin
\[
q := p;
\]
while \( E_2(q) \neq \{\Omega\}, \varnothing \)
\[
do
\]
\[
q := E_2(q);
\]
\[
Q(p) := q
\end.

Obviously, \( \Pi(r) = Q(E_1(r)) \).

We can now show that each while-loop query in the powerset algebra can be expressed in the extended powerset algebra.

**Lemma 6.3.2.** A while-loop query in the powerset algebra can be expressed in the extended powerset algebra.

**Proof.** Let \( r = (\Omega, \omega) \) be a relation. Reconsider the while-loop query \( Q \) of Definition 6.2.1, and suppose that \( E_1(x) \) and \( E_2(x) \) are pae\(s. \) We will construct an epae equivalent to \( Q. \) If \( \omega = ?_\Omega \), then \( Q(r) = (\Omega, ?_\Omega) \) and hence every epae returning relations with an appropriate scheme will return the correct answer in this case. Therefore, we assume in the remainder of this proof that \( \omega \neq ?_\Omega \).
Suppose, without loss of generality, that \( \Omega \) and \( \Omega^{E_1} \) have no atomic attributes in common. Let \( \varphi \) be a one to one mapping from \( U \) to itself such that no atomic attribute of \( \Omega \) and \( \Omega^{E_1} \) is contained in the range of \( \varphi \). Let \( \Omega^\varphi = \varphi(\Omega) \) and \( \Omega^{E_1}_\varphi = \varphi(\Omega^{E_1}) \).

Now let \( p \) be a relation over the scheme \( \{\Omega\} \) and consider the following if–then–else query \( Q' \) of type \( \{\Omega\} \):

\[
\begin{align*}
\text{begin} \\
\text{if } p \times E_1 M(p) \neq (\{\Omega\} \cup \Omega^{E_1}, \emptyset) \\
\text{then} \\
Q'(p) := NE_2 M(p) \\
\text{else} \\
Q'(p) := (\{\Omega\}, \emptyset) \\
\text{end.}
\end{align*}
\]

Let \( E_3(y) \) be a pae expressing \( Q' \) (Theorem 5.1). Obviously, \( E_3 N(r) \) is empty if and only if \( E_3(r) \) is empty. Furthermore, \( E_3(\{\Omega\}, \emptyset) \) is empty.

We now introduce the following expression, defined on relations with scheme \( \{\{\Omega\}\} \), using the notation introduced in Definition 4.5:

\[ E_4(x) = z \cup \rho_{\{\Omega\} - \{\Omega^\varphi\}} \pi_{\{\Omega^\varphi\}} \overline{E_3}(z). \]

This expression is clearly scheme preserving and increasing. Since \( \overline{E_3}(z) \) is basically defined tuple-wise, it is also monotone. Hence we may consider the following expression, defined on relations with scheme \( \Omega \):

\[ E_5(x) = E_4^* N^2(x). \]

If existing, let \( k \geq 1 \) be the smallest integer for which \( E_1 E_2^k - 1(r) = (\Omega^{E_1}, \emptyset) \). (Otherwise, put \( k = \infty. \)) Then \( E_5(r) \) is a relation with scheme \( \{\{\Omega\}\} \) consisting of all relations \( NE_2^i(r) \) with \( 0 \leq i < k \), augmented with the empty relation over \( \{\Omega\} \) if \( k < \infty \). Now consider, again using the notations of Definition 4.5:

\[ E_6(x) = \pi_{\{\{\Omega\}\}} \sigma_{\Omega^{E_1}_\varphi} = \varphi E_1 M E_5(x) \]

We invite the reader to check that selection on emptiness can be expressed in the nested algebra.

We now consider three cases:

- Case 1. \( k < \infty \) and \( E_1(\Omega, \emptyset) \neq (\Omega^{E_1}, \emptyset) \). Then \( E_6(r) \) only consists of \( NE_2^k - 1(r) \), whence \( E_7(r) = M v M E_6(r) = E_2^k - 1(r) \), the result of the while-loop query.

- Case 2. \( k < \infty \) and \( E_1(\Omega, \emptyset) = (\Omega^{E_1}, \emptyset) \). Then \( E_6(r) \) consists of \( NE_2^k - 1(r) \) and \( (\{\Omega\}, \emptyset) \), whence \( M v M E_6(r) = NE_2^k - 1(r) \) and \( E_7(r) = M v M E_6(r) = E_2^k - 1(r) \), the result of the while-loop query.

- Case 3. \( k = \infty \). Then \( E_6(r) \) is empty, whence \( E_7(r) = M v M E_6(r) = (\Omega, \emptyset) \),
the correct result, since \( k = \infty \) is precisely the case in which the while loop runs indefinitely.

Hence we may conclude that \( E_\gamma(x) \) is the desired pae.

**COROLLARY 6.3.1.** An always ending while-loop query in the powerset algebra is equivalent to a pae.

*Proof.* If the while-loop query \( Q \) in the proof of Lemma 6.3.2 above can never run indefinitely, then Case 3 cannot occur. Hence we can omit the \( v \)-operator from expression \( E_\gamma(x) \) without affecting the result; this modified expression is obviously a pae.

From Theorem 6.2.1, Lemma 6.3.1, and Lemma 6.3.2, we may now conclude:

**THEOREM 6.3.1.** The while-closures of the nested algebra, the extended nested algebra, the powerset algebra, and the extended powerset algebra are all equivalent to the extended powerset algebra.

7. INTRODUCING FOR-Loops IN THE ALGEBRA

In the beginning of the previous section, we discussed the language \( QL \) introduced in \([4]\). Due to the fact that variables are not ranked, i.e., that the scheme of the relation they represent can grow wider during the computation process, it is possible to simulate counting in \( QL \), which, in combination with the presence of while-loops, gives \( QL \) the power of general Turing machines. Of course we cannot expect the same for the powerset algebra, where all expressions are ranked. However, we still retain some of that, since in this section we show that a certain type of for-loop can be expressed in the powerset algebra.

First, we formally define for-loop queries. We introduce them in more or less the same fashion as while-loop queries.

**DEFINITION 7.1.** Let \( r = (\Omega, \omega) \) be a relation. Let \( E_1(x) \) and \( E_2(x) \) be unary expressions, defined on relations with scheme \( \Omega \), such that \( E_2(x) \) is scheme preserving. A *for-loop query of type* \( \Omega \) is a unary query \( Q \) for which \( Q(r) \) can be written as the result of a program of the following form:

```
begin
s := r;
for i := 1 to |E_1(r)|
do 
s := E_2(s);
od;
Q(r) := s
end.
```
In this program, \(|E_i(r)|\) stands for the number of tuples of \(E_i(r)\). We assume that \(Q(r)\) takes the value \((\Omega, ?_\Omega)\) if \(E_i(r) = (\Omega E_i, ?_{\Omega E_i})\).

If the expressions in Definition 7.1 are all naes (respectively enaes, paes, or epaes), we say that \(Q\) is a for-loop query in the nested algebra (respectively the extended nested algebra, the powerset algebra, or the extended powerset algebra).

**Example 7.1.** Again, let us have a look at the transitive closure of a binary flat relation. Let \(E(x)\) be the expression defined in Example 4.1. Consider the following for-loop query of type \(\{A, B\}:\)

\[
\begin{align*}
\text{begin} \\
\quad s &:= r; \\
\quad \text{for } i := 1 \text{ to } |\pi_{\{A\}}(r) \times \pi_{\{B\}}(r)| \text{ do} \\
\quad &\quad s := E(s); \\
\quad \text{od}; \\
\quad Q(r) &:= s \\
\text{end.}
\end{align*}
\]

Obviously, \(Q(r)\) equals the transitive closure of \(r\). Hence the transitive closure can be expressed using a for-loop query in the nested algebra.

As in the previous section, we can define the for-closures of the nested algebra, the extended nested algebra, the powerset algebra, and the extended powerset algebra. We are now going to show that:

1. The for-closures of the nested algebra and the powerset algebra are both equivalent to the powerset algebra;
2. The for-closures of the extended nested algebra and the extended powerset algebra are both equivalent to the extended powerset algebra.

First, we show that the powerset operator can be expressed using a for-loop query in the nested algebra.

**Lemma 7.1.** The powerset operator can be expressed in the for-closure of the nested algebra.

**Proof.** Let \(r = (\Omega, \omega)\) be a relation. Let \(\phi\) and \(\psi\) be permutations on \(U\) such that \(\Omega, \Omega^\phi = \phi(\Omega),\) and \(\Omega^\psi = \psi(\Omega)\) have no atomic attributes in common. Recall the expressions

\[
E_1(x) = \pi_{\{\Omega\}} v_{\Omega} \sigma_{\Omega - \Omega^\phi}(x \times \rho_{\Omega^\phi - \Omega}(x)) \cup (\{\Omega\}, \{\emptyset\})
\]

and

\[
E_2(y) = \pi_{\{\Omega\}} v_{\Omega} \mu_{\Omega}(\sigma_{\Omega - \Omega^\phi}(y \times \rho_{\Omega^\phi - \Omega}(y) \times \rho_{\Omega^\psi - \Omega}(y))) \\
\cup \sigma_{\Omega - \Omega^\psi}(y \times \rho_{\Omega^\psi - \Omega}(y) \times \rho_{\Omega^\psi - \Omega}(y))) \cup (\{\Omega\}, \{\emptyset\})
\]

introduced in the proof of Lemma 6.3.1.
Now let \( p \) be a relation over the scheme \( \{ \Omega \} \) and consider the following for-loop query \( Q \):

\[
\begin{align*}
\text{begin} \\
q := p; \\
\text{for } i := 1 \text{ to } |p| \\
& \text{do} \\
& \quad q := E_2(q) \\
& \text{od}; \\
Q(p) := q \\
\text{end.}
\end{align*}
\]

As in the proof of Lemma 6.3.1, it is readily seen that \( \Pi(r) = Q(E_1(r)) \), whence the lemma.

In order to complete the proof of our earlier claim, we still need to show:

**Lemma 7.2.** A for-loop query in the powerset algebra (respectively the extended powerset algebra) can be expressed in the powerset algebra (respectively the extended powerset algebra).

**Proof.** Let \( r = (\Omega, \omega) \) be a relation. As in the proof of Lemma 6.3.2, we may suppose that \( \omega \neq ?\omega \). Reconsider the for-loop query \( Q \) of Definition 7.1, and suppose that \( E_1(x) \) and \( E_3(x) \) are paes (respectively epaes). We show that \( Q \) can be expressed in the while-closure of the powerset algebra. Without loss of generality, we assume that \( \Omega \) and \( \Omega^{E_1} \) have no atomic attributes common. Let \( \varphi \) be a permutation on \( U \) such that \( \Omega^{E_1} \) and \( \Omega^{E_1}_\varphi = \varphi(\Omega^{E_1}) \) have no atomic attributes in common. Now consider the following expression, defined on relations with scheme \( \{ \Omega^{E_1} \} \):

\[
E_3(y) = \pi_{\{\Omega^{E_1}\}} \sigma_{\Omega^{E_1}_\varphi} \sigma_{\Omega^{E_1}_\varphi} \subseteq \varphi^{E_1}(y \times \rho^{E_1}_\varphi - \Omega^{E_1}(y)).
\]

Again, we invite the reader to check that selection with respect to inequality can be expressed in the nested algebra.

Let \( p \) be a relation over \( \{\Omega^{E_1}\} \). Since such a relation can be interpreted as a set of relations over \( \Omega^{E_1} \), \( E_3(p) \) can be obtained out of \( p \) by deleting from it the minimal relations with respect to inclusion. In particular, it follows that \( |E_1(r)| \) is the smallest integer \( i \) for which \( E_1'(\Pi E_1(r) - (\{\Omega^{E_1}\}, \{\emptyset\})) = (\{\Omega^{E_1}\}, \emptyset) \). Now let \( r' \) be a relation over the scheme \( \Omega \cup \{\{\Omega^{E_1}\}\} \) and consider the following while-loop query \( Q' \) of type \( \Omega \cup \{\{\Omega^{E_1}\}\} \) in the powerset algebra:

\[
\begin{align*}
\text{begin} \\
s' := r'; \\
\text{while } M\pi_{\{\Omega^{E_1}\}}(s') \neq (\{\Omega^{E_1}\}, \emptyset) \\
& \text{do} \\
& \quad s' := E_2\pi_{\Omega}(s') \times NE_3 M\pi_{\{\Omega^{E_1}\}}(s') \\
& \text{od}; \\
Q'(r') := s' \\
\text{end.}
\end{align*}
\]
In the above while-loop query, we used the global nest operator to prevent losing the actual result of our computation when the test relation becomes empty. By Theorem 6.3.1, there exists an epae \( E_4(z) \) equivalent to \( Q' \). Moreover, when \( E_1(x) \) and \( E_2(x) \) are both paes (and hence also \( E_3(y) \)), we may assume by Corollary 6.3.1 that \( E_4(z) \) is also a pae. Finally, we have that

\[
Q(r) = \pi_\varnothing E_4(r \times N(IIE_4(r) - (\{\Omega^{E_1}\}, \{\emptyset\}))),
\]

whence the lemma.

By Lemma 7.1 and 7.2, we may conclude

**Theorem 7.1.** The for-closures of both the nested algebra (respectively the extended nested algebra) and the powerset algebra (respectively the extended powerset algebra) are equivalent to the powerset algebra (respectively the extended powerset algebra).

**8. Conclusion**

In this paper, we discussed various ways to enrich the expressive power of the nested algebra, using least fixpoint and programming constructs, inspired by [4, 5]. We showed that the introduction of these constructs yields query languages having the same expressiveness as (a slight extension of) the powerset algebra, thus underlining both the richness of the powerset algebra and its naturalness as a standard to measure the expressive power of query languages for nested relations. However, our results are all heavily language-dependent. In [13, 14], Hull and Su show that the powerset algebra can express exactly the generic queries computable in hyperexponential time (space). Maybe this result can serve as a language-independent criterion for the naturalness of the powerset algebra.

Also, the results in this article indicate possible implementations of the powerset algebra without having to use the much too expensive powerset operator. Indeed, from the results in Section 6, it is readily seen that \( NRQL \), the natural extension to nested relations of the "flat" query \( RQL \) of Chandra and Harel [5], has the same expressiveness as the extended powerset algebra.

**Acknowledgments**

We thank Jan Paredaens for helpful and inspiring discussions on some issues in this article, Jan Van den Bussche for his careful reading of an earlier version, and the referee for his useful comments that helped us to improve the paper. The first author also acknowledges the financial support of IBM Belgium, which enabled him to visit Indiana University, where this joint research was performed.
REFERENCES


