University of Glasgow
Department of Computing Science

Lilybank Gardens
Glasgow G12 8QQ

University of St Andrews
Department of Computational Science

North Haugh
St. Andrews KY16 8SX

A Domain Theoretic Approach to Higher-Order Relations

Persistent Programming
Research Project 28

Alan Dewhe.

A Domain Theoretic Approach to Higher-Order Relations

Peter Buneman†

University of Pennsylvania
Philadelphia, PA 19104

Many database programming languages [Atki85] share with relational database theory the constraint that relations (or whatever bulk data structures are used) should be *flat*. This means that the values stored in a relation may only belong to the base types of the language, integer, boolean, string etc. and may not be compound types such as array, record etc. In relational database theory one usually makes the *first normal form* assumption [Ullm82], which similarly demands that values stored in a a relation be atomic, i.e. they cannot be decomposed by operators of the relational calculus or algebra.

The flatness constraint in database programming languages is rather annoying because, unlike array types and record types, relation types cannot be freely parameterized by other types. This is extremely limiting in the case that relations are the only persistent type allowed for one then has no method of storing arrays, for example, in the database. The flatness constraint also limits the development of database adjuncts for programming languages like Ada [Ichb79] and ML[Gord79] with generic or polymorphic type systems because relations cannot be made "first class" types. In particular one cannot write generic code for data types with this limitation. The first normal form assumption for relational database theory is also restrictive in that it does not allow the relational data model to be cleanly combined with other data models such as the functional data model [Ship81, Bune82]. Moreover it has recently been argued [Banc85, Zani84] that this constraint is incompatible with representation of databases in logic programming and with the requirements of various kinds of application.

In this paper I want to show that by exploiting a form of inheritance on objects rather than types, one can naturally provide a unifying framework for records, relations and other data types that are common in databases and at the same time relax the constraint that they are flat. Moreover, several of the basic ideas of relational database theory have a remarkably simple characterization within this framework. The concept of inheritance has been around for some time in programming languages [Gold80], databases [Smit77] and AI [Brac85]. Only recently has it received a formal treatment in the context of type systems

† Much of this work was carried out while I was a visiting research fellow at the University of Glasgow. I am grateful to the British Science and Engineering Research Council, The University of Pennsylvania and Dr. A.T.E. Matonis for financial support.

in functional programming [Card84, Card85] and in relationship to logic programming AïtK84]. The ideas presented here are based on this work and are also closely related to those described in [Banc85].

The structure of what follows is first to introduce the notion of inheritance informally, the following section then provides a formal description of this in simple domain-theoretic terms and provides a generalized definition of relations. The final two sections show how some of the basic ideas of database theory, such as functional dependencies fit with this description.

Preliminaries

Our starting point is to describe partial functions that behave properly with respect to inheritance. To give some motivation for this, let us introduce a provisional notation for partial functions. The expression $\{'Susan' \Rightarrow 3490; 'Peter' \Rightarrow 7731; 'Karen' \Rightarrow 8535\}$ describes a small telephone directory, a partial function from character strings to integers. We could use the same notation to describe records such as $\{Name \Rightarrow 'J.\ Doe'; Department \Rightarrow 'Sales'; ShoeSize \Rightarrow 10\}$. Here the inputs are a set of labels (perhaps a subset of some larger set) and the outputs are a set of heterogeneous values. Later in this exposition, we shall want to describe partial functions whose output set consists of one understood value. Let us use the notation $\{2; 3; 5; 7\}$ to describe the partial function $\{2 \Rightarrow \{\};3 \Rightarrow \{\};5 \Rightarrow \{\};7 \Rightarrow \{\}\}$, where $\{\}$ is the single output value. There is an obvious, though somewhat misleading as it turns out, correspondence between such single-valued partial functions and sets.

There is a sense in which we can say one partial function is better defined than another. For example $\{'Susan' \Rightarrow 3490; 'Peter' \Rightarrow 7731; 'Karen' \Rightarrow 8535\}$ is better defined than $\{'Susan' \Rightarrow 3490; 'Karen' \Rightarrow 8535\}$ because it is defined for more input values; moreover, wherever the second partial function is defined, it agrees with the first. Another way in which one partial function may be better defined than another is by being defined on the same values and having better defined outputs. For example

$$\{Name \Rightarrow 'John\ Doe'; Address \Rightarrow \{City \Rightarrow 'Philadelphia'; Zip \Rightarrow 19101\}\}$$

is better defined than

$$\{Name \Rightarrow 'John\ Doe'; Address \Rightarrow \{City \Rightarrow 'Philadelphia'\}\}$$

because one of the outputs is better defined in the first expression than in the second. The last two examples are "higher order" partial functions: the values are themselves partial functions.

A problem arises when we use higher order partial functions. We have to ask whether any partial function we write down in this notation makes sense. Compare the following three examples

$$\begin{array}{l} \{\{Emp\# \Rightarrow 1234 \qquad\qquad\quad\ \} \Rightarrow \{Name \Rightarrow 'J.\ Brown'; \ \ Office \Rightarrow 'Paris'\}; \\ \ \{Emp\# \Rightarrow 1234; \ ShoeSize \Rightarrow 10\} \Rightarrow \{Name \Rightarrow 'K.\ Smith' \qquad\quad\ \}\} \end{array} \quad (a)$$

$$\begin{array}{l} \{\{Stud\# \Rightarrow 3456 \qquad\qquad\qquad\quad\ \} \Rightarrow \{Name \Rightarrow 'D.\ Dare'\ \}; \\ \ \{\qquad\qquad\ Course\# \Rightarrow 'CIS123'\} \Rightarrow \{CName \Rightarrow 'Algebra'\ \}; \\ \ \{Stud\# \Rightarrow 3456; \ Course\# \Rightarrow 'CIS123'\} \Rightarrow \{Name \Rightarrow 'D.\ Dare'; \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ CName \Rightarrow 'Algebra'; \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ Grade \Rightarrow 'A' \qquad\ \}\} \end{array} \quad (b)$$

$$\begin{array}{l} \{\{Emp\# \Rightarrow 1234 \qquad\qquad\quad\ \} \Rightarrow \{Name \Rightarrow 'J.\ Brown'; \ \ Office \Rightarrow 'London'\}; \\ \ \{Emp\# \Rightarrow 1234; \ ShoeSize \Rightarrow 10\} \Rightarrow \{Name \Rightarrow 'J.\ Brown'; \ \ Office \Rightarrow 'London'\}; \\ \ \{Emp\# \Rightarrow 1234 \qquad\qquad\quad\ \} \Rightarrow \{Name \Rightarrow 'J.\ Brown' \qquad\qquad\quad\ \}\} \end{array} \quad (c)$$

Example (a) is badly behaved. In return for a better input it has produced a less informative - and contradictory - output. Example (b) is the sort of behavior one might expect from a database system. There is extra information to be gained by providing a better specified input. Example (c) is redundant in that we can infer the second and third input-output pairs from the first, but we can nevertheless consider these pairs as part of the partial function. In order to exclude partial functions like (a) we need to impose the condition that if a given input x produces an output y, then any input that is better defined than x will produce an output that is at least as well-defined as y. Looking at example (c) we see something that is not even a partial function; a given input $\{Emp\# \Rightarrow 1234\}$ has produced two outputs. Given that there is an ordering on the input and output spaces, we need to define formally what we mean by a partial function.

Maps, Records and Relations

The examples in the previous section were mostly composed of records, i.e. partial functions from labels to values. In order to define formally what it means for a partial function to be well behaved, we shall assume the input and output spaces are partially ordered. Specifically, we shall assume that they are complete partial orders (c.p.o.s). If V and W are the input and output c.p.o.s, we can define a partial function as a subset F of $V \times W$ that is subject to the following restrictions.

(1) if $(x, y) \in F, x' \sqsupseteq x$ and $y \sqsupseteq y'$ then $(x', y') \in F$

(2) if $(x, y_1) \in F$ and $(x, y_2) \in F$ then $(x, y_1 \sqcup y_2) \in F$

(3) $(\perp_1, \perp_2) \in F$

The first of these says that if a given input-output pair belongs to F then any better defined input and worse defined output will belong to F. The second says that F is *functional*, i.e. there is a unique best output associated with a given input. (3) indicates that every input in V has at last one associated output (namely $\perp_2$). Since a given input can have more than one output, it is inappropriate to call F a function (partial or total). Instead, we shall refer to any subset of $V \times W$ that satisfies these conditions as a *map* from V to W and denote the set of such maps by $V \mapsto W$. It is interesting to note that the conditions for a map are almost the same as those used by Scott [Scot82] for an *approximable function* in his "Information Systems" approach to denotational semantics, and work in progress by Ohori [Ohor86] promises to provide a proper denotational semantics for database theory.

Now it is readily seen that maps are ordered by (set) inclusion and that if F_1 and F_2 are in $V \mapsto W$ then $F_1 \cap F_2$ is also in F. We can therefore define

$$F_1 \sqcap F_2 = F_1 \cap F_2$$

$$F_1 \sqcup F_2 = \cap\{F | F \in V \mapsto W,\ F \supseteq F_1,\ F \supseteq F_2\}$$

The join ($\sqcup$) is defined only if there is at least one map F such that $F \supseteq F_1$ and $F \supseteq F_2$. Thus maps themselves form a c.p.o. Moreover, if $F \in V_1 \mapsto V_2$ and $G \in V_2 \mapsto V_3$ their composition, defined by $F_1 \circ F_2 = \{(x, z) | \exists y.(x, y) \in F_1,\ (y, z) \in F_2\}$, is a map. For maps in $V \mapsto V$ the ordering that defines V (note that it is a map) is the identity for composition.

A particularly simple map is that specified by a single pair of points (x, y), i.e. $\{x', y' | x' \in V, y' \in W, x' \sqsupseteq x, y \sqsupseteq y'\}$. Call such a map *elementary* and call a map *finite* if it is of the form $E_1 \sqcup E_2 \sqcup \ldots \sqcup E_n$ where $E_1, E_2, \ldots, E_n$ are elementary. If F_1 and F_2 are finite maps in $V \mapsto W$, then so are $F_1 \sqcap F_2$ and $F_1 \sqcup F_2$ (if it exists). If at least one of F_1 and F_2 is finite, then so is $F_1 \circ F2$.

We are now in a position to be more precise about the notation of the previous section. The notation $\{x_1 \Rightarrow y_1; x_2 \Rightarrow y_2; \ldots; x_n \Rightarrow y_n\}$ defines the finite map which is the join ($\sqcup$) of the n elementary maps generated by the pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Note that this join is not always defined, which is demonstrated by example (a) of the previous section.

All we have done so far is to re-hash some of the basic definitions of domain theory [Stoy77] in a somewhat unusual manner. Let us try to see how this is connected to databases by taking some examples.

1. Suppose V is a flat domain and $TRIV$ is the trivial domain which contains one non-bottom element $\{\}$. Now the elements of $V \mapsto TRIV$ correspond to the subsets of elements of V (other than $\perp$). The $\sqcup$ and $\sqcap$ operations correspond to union and intersection, and the ordering on $V \mapsto TRIV$ corresponds to set inclusion.
2. Now suppose $\mathcal{L}$ is a flat domain of *labels* and V is any domain. We can identify $\mathcal{L} \mapsto V$ with the *records* over V. The ordering on $\mathcal{L} \mapsto V$ is the ordering we used informally in the previous section. Call this domain $\mathcal{R}(V)$. $\sqcup$ and $\sqcap$ respectively define "unifiers" and "generalizers" of two records. Note that for records r_1 and r_2, $r_1 \sqcup r_2$ is not necessarily defined.
3. We can now consider the maps in $\mathcal{R}(V) \mapsto TRIV$. These are the *relations* over V, with $\sqcap$ defining the *natural join*. Note that there is a difference between relations and sets as defined in (1) above. The input domain $\mathcal{R}(V)$ is not, in general, flat.

This result that natural join is a "meet" operation ($\sqcap$) is not so surprising if one considers two relations with the same columns. Here the natural join gives us the intersection of the tuples in the two relations. What is more interesting is that this definition extends to "non-first-normal-form" relations, where the components may themselves be structures such as records or other relations. For example, the natural join of

$$\begin{array}{l}\{\{Name \Rightarrow \text{'J. Doe'};\ Dept \Rightarrow \text{'Sales'};\ Addr \Rightarrow \{City \Rightarrow \text{'Moose'} \quad\}\};\\ \ \{Name \Rightarrow \text{'M. Dee'};\ Dept \Rightarrow \text{'Manuf'} \quad\};\\ \ \{Name \Rightarrow \text{'N. Bug'};\ Addr \Rightarrow \{ \quad State \Rightarrow MT\}\}\end{array}$$

and

$$\begin{array}{l}\{\{Dept \Rightarrow \text{'Sales'};\ Addr \Rightarrow \{ \quad State \Rightarrow WY\ \}\};\\ \ \{Dept \Rightarrow \text{'Admin'};\ Addr \Rightarrow \{City \Rightarrow \text{'Billings'} \quad\}\};\\ \ \{Dept \Rightarrow \text{'Manuf'};\ Addr \Rightarrow \{ \quad State \Rightarrow MT\ \}\}\}\end{array}$$

is

$$\begin{array}{l}\{\{Name \Rightarrow \text{'J. Doe'};\ Dept \Rightarrow \text{'Sales'};\ Addr \Rightarrow \{City \Rightarrow \text{'Moose'};\ State \Rightarrow WY\}\};\\ \ \{Name \Rightarrow \text{'M. Dee'};\ Dept \Rightarrow \text{'Manuf'};\ Addr \Rightarrow \{ \quad State \Rightarrow MT\}\};\\ \ \{Name \Rightarrow \text{'N. Bug'};\ Dept \Rightarrow \text{'Manuf'};\ Addr \Rightarrow \{ \quad State \Rightarrow MT\}\};\\ \ \{Name \Rightarrow \text{'N. Bug'};\ Dept \Rightarrow \text{'Admin'};\ Addr \Rightarrow \{City \Rightarrow \text{'Billings'};\ State \Rightarrow MT\}\}\}\end{array}$$

We identified relations with maps from $\mathcal{R}(V)$ to a trivial domain because it corresponds to our intuition that a relation is something like a set of records, and we would need to keep this characterization if we wanted to assign data types to relations. However, in the following discussion of functional dependencies, we do not need to be so restrictive. We can identify relations with maps in $V \mapsto TRIV$ for any domain V. We shall occasionally use the ordering on records to provide some examples.

Functional Dependencies

Having found a simple characterization of the natural join, can we do the same for other relational operations such as projection; and how do we characterize functional dependencies and other concepts in relational database theory using this domain-theoretic approach? Our observation that a relation is an element of $V \mapsto TRIV$ is equivalent to saying that relations are upward-closed subsets of V. (A subset S of V is *upward-closed* if $x \in S$, $x' \in V$ and $x' \sqsupseteq x$ imply $x' \in S$). The natural join is the intersection of such sets, which itself is upward closed. Now if we are dealing with finite maps we can characterize such sets by their minimal elements. Suppose R and S are two such sets of minimal elements and $\overline{R}$ and $\overline{S}$ are their respective upward closures. The ordering on $V \mapsto TRIV$ is the containment ordering on upward closed subsets, and

$$\overline{R} \subseteq \overline{S} \quad \text{iff} \quad \forall s \in S. \exists r. r \in R, s \sqsupseteq r.$$

The natural join of two relations is given by the intersection, and

$$\overline{T} = \overline{R} \cup \overline{S} \quad \text{iff} \quad T = \mathit{minset}\{r \sqcup s | r \in R, s \in S\}.$$

Where $\mathit{minset}(S)$ is the set of minimal elements of a set S. When we are talking about the joins of relations defined by their minimal representatives, we shall use the notation $R \bowtie S$ i.e. $R \bowtie S = \mathit{minset}(\overline{R} \cup \overline{S})$. Note that these minimal elements are pairwise incomparable (i.e. they form cochains). Also note that Aït-Kaci uses the $\bowtie$ operation (as defined here) to provide semantics for his interpreter for Login.

Now in order to talk about functional dependencies and projections we need to place a slightly stronger constraint on what constitutes these sets of minimal elements. We shall require, for reasons that will shortly become apparent, that they be pairwise inconsistent. David S. Warren also requires this condition for the data base component of his logic programming system. We shall call a set of pairwise inconsistent elements *independent*; and if V is a c.p.o. we shall use $I(V)$ to denote the set of independent sets in V.

Prop 1. *If V is a c.p.o. then $I(V)$ is a c.p.o. under the ordering $\leq$ defined by $R \leq S$ iff $\overline{S} \subseteq \overline{R}$ with $\bowtie$ being the lub operation.*

The proof that any two members I_1 and I_2 have a *lub* is immediate. That they have a *glb* follows by considering, for each $v \in I_1 \cup I_2$, the *lubs* of the set

$$\{w | w \sqsubseteq v \text{ and } \exists I \in I(V). I \leq I_1, I \leq I_2 \text{ and } w \in I\}.$$

It is an immediate consequence of this result that $I_1 \bowtie I_2$ is an independent set whenever I_1 and I_2 are independent. It is also something of a relief that "natural joins"

are once again "joins" under the ordering $\leq$ (they were "meets" under the ordering we defined on $V \mapsto TRIV$.)

The usual way to think of a *functional dependency* is as a constraint on a relation. We shall adopt a slightly different approach and think of a functional dependency as a map induced by a relation. Of course, some such maps will be "accidental" and must later be ruled out since they are not part of the intended structure of the relation. However, this approach will allow us to characterize relationships among functional dependencies in a particularly simple way. Suppose R is a member of $I(V)$. Just as we defined $\overline{R}$ to be the set of points in V *above* points in R, we can define $\underline{R} = \{x | \exists r. r \in R, r \sqsupseteq x\}$, the set of points *below* points in R. Roughly speaking, we can say that there is a functional dependency from A to B if R induces a map from $\overline{A}$ to $\underline{B}$, where A and B are in $I(V)$. But we usually think of functional dependencies as being defined on sets of column names. What do these have to do with independent sets? Think of the set of all records whose *Name* and *Age* fields are defined to the point that they are character strings and integers respectively. Take the set of minimal members of this set. It is clearly a set of pairwise inconsistent records, hence independent; and we may think of it as characterizing the data type $\{\mathit{Age}\text{: }\mathit{string}; \mathit{Name}\text{: }\mathit{int}\}$. In fact we can define

$$\{\mathit{Name}\text{: }\mathit{string};\ \mathit{Age}\text{: }\mathit{int}\} = \{\{\mathit{Name} \Rightarrow s; \mathit{Age} \Rightarrow i\} | s \in \mathit{string}, i \in \mathit{int}\}.$$

We shall occasionally refer to such sets as *types* although there is no formal distinction between types and relations; they are both independent sets. No immediate connection is intended with the definition of *type* given by programming language semanticists in, for example, [MacQ82]. However in [Bune85] a simple type system for relations is described based upon this domain theoretic approach, which appears to combine cleanly with Cardelli's work on multiple inheritance [Card84]. Although our example of $\{\mathit{Name}\text{: }\mathit{string}; \mathit{Age}\text{: }\mathit{int}\}$ is an example of a flat " record" type, there are more interesting independent sets such as those defined by the set of records that have an *Address* whose *City* is defined or the set of records whose *Age* and *ShoeSize* are defined and equal. Note that the set of records for which *either* the *Name* or the *Age* is defined does not give rise to an independent set. We must not therefore expect to be able to define functional dependencies for such disjunctive types.

To be precise about how relations generate functional dependencies, suppose A, B, R are sets in $I(V)$ with $A \leq R$. Consider the set of pairs

$$\{(a,b) | a \in \overline{A}, b \in \underline{B}, \exists r \in \overline{R}. r \sqsupseteq a \sqcup b\} \cup \{(x, \bot) | x \in V\}. \qquad \text{(A)}$$

If this is a map, we shall say that $(A, B) \in R^+$ or, in database parlance, R satisfies a functional dependency from A to B. What this means is that if the pairs of points in $\overline{A}$ and

$\underline{B}$ that are bounded above by some point in $\overline{R}$ form a map, then R "generates" a function from $\overline{A}$ to $\underline{B}$, i.e. R satisfies the functional dependency (A, B). The term $\{(x, \bot)|x \in V\}$ is added to take care of the constraint that any map must contain $(\bot, \bot)$. To give an example to justify the claim that this is a functional dependency, let A be the set of points with a defined *Name* (e.g. $\{Name: string\}$, and B the set of points with a defined *Age* (e.g. $\{Age: int\}$). The relation

$$\begin{array}{lll} \{\{Name \Rightarrow 'J.Doe'; & Dept \Rightarrow 'Sales'; & Age \Rightarrow 21\}; \\ \{Name \Rightarrow 'M.Mack'; & Dept \Rightarrow 'Manuf' & \}; \\ \{Name \Rightarrow 'N.Bug'; & & Age \Rightarrow 21\}\} \end{array}$$

will induce a map (according to the above definition) from $\overline{\{Name: string\}}$ to $\underline{\{Age: int\}}$, namely $\{\{Name \Rightarrow 'J.\ Doe'\} \Rightarrow \{Age \Rightarrow 21\}; \{Name \Rightarrow 'N.\ Bug'\} \Rightarrow \{Age \Rightarrow 21\}\}$, but the relation

$$\begin{array}{lll} \{\{Name \Rightarrow 'J.Doe'; & Dept \Rightarrow 'Sales'; & Age \Rightarrow 21\}; \\ \{Name \Rightarrow 'M.Mack'; & Dept \Rightarrow 'Manuf' & \}; \\ \{Name \Rightarrow 'J.Doe'; & & Age \Rightarrow 22\}\} \end{array}$$

will not.

Another way of thinking about functional dependencies is that R satisfies a dependency from A to B if B gives us no more information about R than A, i.e. that members of R that are not discriminated by A will not be discriminated by B. Stated mathematically,

Prop 2. *$(A, B) \in R^+$ iff for any $r_1, r_2 \in R$, $a \in A$, $b \in B$, if $r_1 \sqsubseteq a, r_2 \sqsubseteq a$ and $r_1 \sqsubseteq b$ then $r_2 \sqsubseteq b$.*

In the case that $A \leq R$ and $B \leq R$, this means that for $(A, B) \in R^+$ the partition of R induced by B must be coarser than the partition of R induced by A. There is obviously a relationship here with the partition semantics for relations described in [Cosm85]; this characterization of functional dependencies is also the basis for a form of denotational semantics for relational databases being developed by Ohori [Ohor86].

Now there are various ways of characterizing the structure of R^+. First, R^+ is a subset of $I(V) \times I(V)$ and is a map with some additional properties, which are given in the following result.

Prop 3. *If $R \in I(V)$ then R^+ is a map (i.e. $R^+ \in I(v) \mapsto I(v)$). Moreover, for all $A, B, C \in I(V)$,*

if $A \geq B$ then $(A, B) \in R^+$, and

if $(A, B) \in R^+$ and $(B, C) \in R^+$ then $(A, C) \in R^+$.

Not surprisingly, these conditions are direct generalizations of Armstrong's axioms for functional dependencies, where we generalize the subset ordering on column names to the ordering $\leq$ in independent sets. The following result, which applies to any c.p.o. V, states this formally.

Prop 4. *For any $F \in V \times V$ and $A, B, C, W \in V$ the following two sets of conditions are equivalent:*

a_1	*F is a map*
b_1	*if $A \sqsupseteq B$ then $(A, B) \in F$*
c_1	*if $(A, B) \in F$ and $(B, C) \in F$ then $(A, C) \in F$*
a_2	*if $A \sqsupseteq B$ then $(A, B) \in F$*
b_2	*if $(A, B) \in F$ and $(B, C) \in F$ then $(A, C) \in F$*
c_2	*if $W \in V$ and $(A, B) \in F$ then $(A \sqcup W, B \sqcup W) \in F$*

It is an immediate consequence of this result that R^+ satisfies Armstrong's axioms as defined in (a_2), (b_2) and (c_2) above.

It was noted earlier that our definitions concerning maps were simply re-statements of some standard results in lattice theory. To be specific, for any F in $V \mapsto W$ we can associate a *monotone function* from V to W by defining, for any $x \in V$, the function $F(x) = \sqcup\{y \in W|(x, y) \in F\}$. We shall briefly use this overloaded notation to describe both the map F and the function F.

Prop 5. *F satisfies Armstrong's axioms iff the associated function from V to V is a closure. That is, for all x in V*

$$F(x) \sqsupseteq x, \text{ and}$$

$$F(F(x)) = F(x).$$

Now a closure is uniquely characterized its set of fixed points, i.e. those values $x \in V$ for which $F(x) = x$. Moreover, the set of such points must be a meet-closed subset of V, i.e. if x_1 and x_2 are fixed-points of F then so is $x_1 \sqcap x_2$. Let us redirect our attention to $I(V)$. We have seen that any relation (independent set) in V generates a map R^+ in $I(V) \mapsto I(V)$ which satisfies a generalized form of Armstrong's axioms. However, Armstrong's axioms are intended to describe *constraints* on a relation, not the relation itself. What this means

is that given a map Q in $I(V) \mapsto I(V)$ which satisfies Armstrong's axioms or, equivalently, given a closure on $I(V)$, a relation R *satisfies* Q iff $R^+ \sqsupseteq Q$, where we take the ordering on maps to be that defined earlier.

There are some consequences of this connection with database theory that are quite straightforward. For example, if R satisfies Q then so does any relation that is above R in the ordering on $I(V)$. In particular, if R satisfies Q then so does $R \bowtie S$. Also, restricting our attention to first-normal-form relations, if we are given a set Q of pairs of subsets of column-names that satisfy Armstrong's axioms, we can characterize Q by the fixed points of the associated closure. These are the sets of column names that are not expanded by any pair in Q, i.e. a set C of column names is is a fixed-point of Q iff for no $C' \geq C$ is (C, C') in Q. To take a standard example [Ullm82] of the relation scheme C(ity), S(treet), Z(ip), with functional dependencies generated by $Z \rightarrow C$ and $\{C, S\} \rightarrow Z$, the fixed points are $\{C, S, Z\}$, $\{C\}$, $\{S\}$.

The observation that maps satisfying Armstrong's Axioms correspond to closures has been made by [Simo85]. The results above show how they generalize to non first-normal-form relations. The completeness and consistency of Armstrong's axioms in this more general case is immediate.

Projections

Unlike the natural join, which gives rise to "higher" relations in the lattice $I(V)$ of independent sets, we would expect the projection operator to loose information and to produce results that are "lower" in $I(V)$. Suppose A is what we have been informally calling a type, e.g. $\{Name$: *string*; *Age*: *int*$\}$ and R is a relation such that $R \sqsupseteq A$. We could very simply define the projection A' of R onto A as the set of points in A below some point in R, i.e. $A' = \{a \in A | \exists r \in R. a \sqsubseteq r\}$. By this definition $A' \in I(V)$ and $A' \leq R$. However, this definition is not very satisfactory because of the constraint that R must be above A. If, for example, R were to have some null values for *Age* this would not be the case.

To produce a more satisfactory definition of projection, one approach is to go back to our definition of R^+ (equation A of the previous section) and note that for any $A \in I(V)$, the pair (R, A) is always in R^+. Stated informally, no independent set A can give us more information about R than R itself. We may therefore ask for the image of R under the map induced from $\overline{R}$ to $\underline{A}$, and call this the projection of R onto A. one way of characterizing this set is to introduce a new ordering, $\preceq$ defined by

$$R \preceq S \quad \text{iff} \quad \forall r \in R. \exists s \in S. r \sqsubseteq S.$$

There is an obvious symmetry between $\preceq$ and $\leq$, and $R \preceq S$ iff $\underline{R} \subseteq \underline{S}$. The projection of A on R is characterized by the following result:

Prop 6. *A' is the image of R under the map induced by (A, R) iff $A' = maxset(\underline{A} \cup \underline{R})$.*

maxset is defined analogously to *minset* in the previous section. In fact, the projection of R on A is nothing more than the *glb* of R and A under the ordering $\preceq$, and projection is a *symmetrical* operation on R and A. Unfortunately, this still does not give us exactly what we want, because this *glb* may not be an independent set. This is readily seen by trying to project

$$\begin{array}{l} \{\{Name \Rightarrow \text{'J. Doe'}; \quad Dept \Rightarrow \text{'Sales'}; \; Age \Rightarrow 21\}; \\ \; \{Name \Rightarrow \text{'M. Mack'}; \; Dept \Rightarrow \text{'Manuf'} \qquad \}; \\ \; \{Name \Rightarrow \text{'N. Bug'}; \qquad\qquad\qquad Age \Rightarrow 22\}\} \end{array}$$

onto {it Dept: *string*; *Age*:*int*}.

At this point we have two options. We could either decide that projection is not always defined, or we could take the greatest independent set that is less than (under the ordering $\leq$) the set we have just defined. Doing this in the example above yeilds

$$\begin{array}{l} \{\{Dept \Rightarrow \text{'Sales'}; \; Age \Rightarrow 21\}; \\ \; \{Manuf \Rightarrow \text{'Sales'}; \; Age \Rightarrow 22\}\} \end{array}$$

Which option we adopt depends, of course, on a better understanding of the semantics of relations which would of necessity give a precise account of null values. However, if only for the sake of providing some mathematical conclusion to the study of projections, let us adopt the second option. Stating this formally:

Prop 7. *For any $A, B \in I(V)$ there is a unique maximal (under $\leq$) independent set $A \barwedge B$ such that $A \barwedge B \preceq A$ and $A \barwedge B \preceq B$.*

Under this definition of projection $\barwedge$ is a commutative, associative operator that also associates with $\bowtie$, i.e. $(A \barwedge B) \bowtie C = A \barwedge (B \bowtie C)$. We might also ask under what conditions $\barwedge$ "distributes" over $\bowtie$. The following result gives us a sufficient condition.

Prop 8. *For any $A, B, R \in I(V)$, if there exists $W \in I(V)$ such that $W \leq A$, $W \leq B$ and $(W, A) \in R^+$ then $(R \barwedge A) \bowtie (R \barwedge B) = R \barwedge (A \bowtie B)$.*

This is a generalization of the lossless join condition for relational instances [Ullm82]. In order to obtain the lossless join condition for relational schemata one needs to be assured that there are some points above A and B, i.e. that the "types" A and B have some instances. The representation of relational schemata is a problem that is left for further investigation.

Conclusions

I have tried to show that the notion of inheritance leads to a natural representation of the operators of the relational algebra and that some of the basic properties of relational database theory, such as Armstrong's axioms, can be derived from some very simple domain theoretic relationships. If these ideas have any value one would expect to be able to represent other notions in database theory, such as multi-valued dependencies and the universal relation assumption within the same framework. However, given the apparent connection with Scott's "Information Systems", a more pressing need is to work out a proper denotational semantics for relational databases.

In the longer term I hope that it will be possible to use an approach such as this to produce better type systems for database programming languages.

Acknowledgements

The ideas presented here were largely stimulated by Aït-Kaci's notation and semantics for type subsumption and Cardelli's work on multiple inheritance. Malcolm Atkinson's proposals for a relational data type for PS-Algol were also useful in thinking about the connection between relations and maps. I am extremely grateful to David MacQueen and Gopalan Nadathur for correcting my imperfect knowledge of lattice theory and to Atsushi Ohori, who was largely responsible for the "informational" characterization of functional dependencies.

References

[AïtK84] Aït-Kaci, H. "A Lattice Theoretic Approach to Computation based on a Calculus of Partially Ordered Type Structures", PhD. Dissertation, Department of Computer and Information Science, Moore School/D2, University of Pennsylvania, Philadelphia, PA 19104. (1984)

[Atki85] Atkinson, M.P. and Buneman, O.P. "Database Programming Language Design", Technical Report 10-85, University of Pennsylvania.

[Banc85] Bancilhon, F. and Khoshafian, S., "A Calculus for Complex Objects", Technical Report, MCC, Austin Texas, October 1985.

[Brac85] Brachman, R.J. and Schmolze, J.G., " An Overview of the KL-One Knowledge Representation System", Cognitive Science, 9,2, April 1985.

[Bune82] Buneman, P., Frankel, R.E. and Nikhil, R., An Implementation Technique for Database Query Languages, ACM Transactions on Database Management, 7, 2, June 1982

[Bune85] Buneman, O.P., "Data Types for Data Base Programming", Proceedings of the Appin Conference on Data Types and Persistence, Technical Report, Department of Computing, Glasgow University, September, 1985.

[Card84] Cardelli, L., "A semantics of Multiple Inheritance", *Semantics of Data Types* Kahn, G., MacQueen, D.B. and Plotkin, G. (eds), Springer-Verlag, Berlin, June 1984

[Card85] Cardelli, L. and Wegner, P., "On Understanding Types, Data Abstraction, and Polymorphism", Technical Report, Brown University, Aug 1985.

[Cosm85] Cosmadakis, S.S., Kanellakis, P.C., Spyratos, N. "Partition Semantics for Relations", Technical Report, Brown University, December 1985.

[Gold80] Goldstein, I. P. and Bobrow, D. G., "Extending object oriented programming in Smalltalk", Proceedings of the 1980 Lisp Conference, August, 1980.

[Gord79] Gordon, M.J., Milner, A.J.R.G., and Wadsworth, C.P., *Edinburgh LCF*, Springer-Verlag, Lecture Notes in Computer Science, 1979.

[Smit77] Smith, J.M. and Smith, D.C.P., "Database Abstractions - Aggregation and Generalization" ACM Transactions on Database Systems, 2, 2, June 1977.

[Ichb79] Ichbiah *et al.*, Rationale of the Design of the Programming Language Ada, ACM Sigplan Notices, 14, 6, 1979

[MacQ82] MacQueen, D.B. and Sethi, R., "A semantic model of types for applicative

languages", Technical Report, Bell Laboratories, 1982.

[Ohor86] Ohori, A. *Personal Communication*, 1986.

[Scot82] Scott, D. "Domains for Denotational Semantics" ICALP '82, Aarhus, Denmark, 1982.

[Ship81] Shipman, D.W., The Functional Data Model and the Data Language DAPLEX, ACM Transactions on Database Systems, 140-173, 6, 1, March 1981

[Simo85] Jurgensen, H and Simovici, D.A., "Towards an Abstract Theory of Dependency Constraints in Relational Databases", Technical Report 1/85, Department of Mathematics and Computer Science, University of Massachussetts, 1985.

[Stoy77] Stoy, J. *Denotational Semantics: The Scott-Strachey approach to Programming Language Theory*. MIT Press, 1977

[Ullm82] Ullman, J.D., *Principles of Database Systems* (2nd ed.), Computer Science Press, Rockville Md. (1982)

[Zani84] Zaniolo, C. "Prolog: A Database Query Language for All Seasons", Proc. IEEE-ACM International Expert Database Systems Workshop, Kiawah Island, October 1984.

Bibliography

Copies of documents in this list may be obtained by writing to:

The Secretary,
Persistent Programming Research Group,
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ
Scotland.

Books

Davie, A.J.T. & Morrison, R.
"Recursive Descent Compiling", Ellis-Horwood Press (1981).

Atkinson, M.P. (ed.)
"Databases", Pergammon Infotech State of the Art Report, Series 9, No.8, January 1982. (535 pages).

Cole, A.J. & Morrison, R.
"An introduction to programming with S-algol", Cambridge University Press, Cambridge, England, 1982.

Stocker, P.M., Atkinson, M.P. & Grey, P.M.D. (eds.)
"Databases - Role and Structure", Cambridge University Press, Cambridge, England, 1984.

Published Papers

Morrison, R.
"A method of implementing procedure entry and exit in block structured high level languages". Software, Practice and Experience 7, 5 (July 1977), 535-537.

Morrison, R. & Podolski, Z.
"The Graffiti graphics system", Proc. of the DECUS conference, Bath (April 1978), 5-10.

Atkinson, M.P.
"A note on the application of differential files to computer aided design", ACM SIGDA newsletter Summer 1978.

Atkinson, M.P.
"Programming Languages and Databases", Proceedings of the 4th International Conference on Very Large Data Bases, Berlin, (Ed. S.P. Yao), IEEE, Sept. 78, 408-419. (A revised version of this is available from the University of Edinburgh Department of Computer Science (EUCS) as CSR-26-78).

Atkinson, M.P.
"Progress in documentation: Database management systems in library automation and information retrieval", Journal of Documentation Vol.35, No.1, March 1979, 49-91. Available as EUCS departmental report CSR-43-79.

Gunn, H.I.E. & Morrison, R.
"On the implementation of constants", Information Processing Letters 9, 1 (July 1979), 1-4.

Atkinson, M.P.
"Data management for interactive graphics", Proceedings of the Infotech State of the Art Conference, October 1979. Available as EUCS departmental report CSR-51-80.

Atkinson, M.P. (ed.)
"Data design", Infotech State of the Art Report, Series 7, No.4, May 1980.

Morrison, R.
"Low cost computer graphics for micro computers", Software Practice and Experience, 12, 1981, 767-776.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"PS-algol: An Algol with a Persistent Heap", ACM SIGPLAN Notices Vol.17, No. 7, (July 1981) 24-31. Also as EUCS Departmental Report CSR-94-81.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Nepal - the New Edinburgh Persistent Algorithmic Language", in Database, Pergammon Infotech State of the Art Report, Series 9, No.8, 299-318 (January 1982) - also as EUCS Departmental Report CSR-90-81.

Morrison, R.
"S-algol: a simple algol", Computer Bulletin II/31 (March 1982).

Morrison, R.
"The string as a simple data type", Sigplan Notices, Vol.17,3, 46-52, 1982.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Progress with Persistent Programming", presented at CREST course UEA, September 1982, revised in "Databases - Role and Structure", see PPRR-8-84.

Morrison, R.
"Towards simpler programming languages: S-algol", IUCC Bulletin 4, 3 (October 1982), 130-133.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Problems with persistent programming languages", presented at the Workshop on programming languages and database systems, University of Pennsylvania. October 1982. Circulated (revised) in the Workshop proceedings 1983, see PPRR-2-83.

Atkinson, M.P.
"Data management", in Encyclopedia of Computer Science and Engineering 2nd Edition, Ralston & Meek (editors) January 1983. van Nostrand Reinhold.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Algorithms for a Persistent Heap", Software Practice and Experience, Vol.13, No.3, 259-272 (March 1983). Also as EUCS Departmental Report CSR-109-82.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"CMS - A chunk management system", Software Practice and Experience, Vol.13, No.3 (March 1983), 273-285. Also as EUCS Departmental Report CSR-110-82.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Current progress with persistent programming", presented at the DEC workshop on Programming Languages and Databases, Boston, April 1983.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"An approach to persistent programming", The Computer Journal, 1983, Vol.26, No.4, 360-365 - see PPRR-2-83.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"PS-algol a language for persistent programming", 10th Australian Computer Conference, Melbourne, Sept. 1983, 70-79 - see PPRR-2-83.

Morrison, R., Weatherill, M., Podolski, Z. & Bailey, P.J.
"High level language support for 3-dimension graphics", Eurographics Conference Zagreb, North Holand, 7-17, Sept. 1983. (ed. P.J.W. ten Hagen).

Cockshott, W.P., Atkinson, M.P., Chisholm, K.J., Bailey, P.J. & Morrison, R.
"POMS : a persistent object management system", Software Practice and Exerience, Vol.14, No.1, 49-71, January 1984.

Kulkarni, K.G. & Atkinson, M.P.
"Experimenting with the Functional Data Model", in Databases - Role and Structure, Cambridge University Press, Cambridge, England, 1984.

Atkinson, M.P. & Morrison, R.
"Persistent First Class Procedures are Enough", Foundations of Software Technology and Theoretical Computer Science (ed. M. Joseph & R. Shyamasundar) Lecture Notes in Computer Science 181, Springer Verlag, Berlin (1984).

Atkinson, M.P., Bocca, J.B., Elsey, T.J., Fiddian, N.J., Flower, M., Gray, P.M.D. Gray, W.A., Hepp, P.E., Johnson, R.G., Milne, W., Norrie, M.C., Omololu, A.O., Oxborrow, E.A., Shave, M.J.R., Smith, A.M., Stocker, P.M. & Walker, J.
"The Proteus distributed database system", proceedings of the third British National Conference on Databases, (ed. J. Longstaff), BCS Workshop Series, Cambridge University Press, Cambridge, England, (July 1984).

Atkinson, M.P. & Morrison, R.
"Procedures as persistent data objects", ACM TOPLAS 7, 4, 539-559, (Oct. 1985) - see PPRR-9-84.

Morrison, R.,Bailey, P.J., Dearle, A., Brown, P. & Atkinson, M.P.
"The persistent store as an enabling technology for integrated support environments", 8th International Conference on Software Engineering, Imperial College, London (August 1985), 166-172 - see PPRR-15-85.

Atkinson, M.P. & Morrison, R.
"Types, bindings and parameters in a persistent environment", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 1-24 - see PPRR-16-85.

Davie, A.J.T.
"Conditional declarations and pattern matching", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 278-283 - see PPRR-16-85.

Krablin, G.L.
"Building flexible multilevel transactions in a distributed persistent environment, proceedings of Data Types and Persistence Workshop, Appin, August 1985, 86-117 - see PPRR-16-85.

Buneman, O.P.
"Data types for data base programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 291-303 - see PPRR-16-85.

Cockshott, W.P.
"Addressing mechanisms and persistent programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 363-383 - see PPRR-16-85.

Norrie, M.C.
"PS-algol: A user perspective", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 399-410 - see PPRR-16-85.

Owoso, G.O.
"On the need for a Flexible Type System in Persistent Programming Languages", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 423-438 - see PPRR-16-85.

Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. & Dearle, A.
"A persistent graphics facility for the ICL PERQ", Software Practice and Experience, Vol.14, No.3, (1986) - see PPRR-10-84.

Atkinson, M.P. and Morrison R.
"Integrated Persistent Programming Systems", proceedings of the 19th Annual Hawaii International Conference on System Sciences, January 7-10, 1986 (ed. B. D. Shriver), vol IIA, Software, 842-854, Western Periodicals Co., 1300 Rayman St., North Hollywood, Calif. 91605, USA - see PPRR-19-85.

Atkinson, M.P., Morrison, R. and Pratten, G.D.
"A Persistent Information Space Architecture", proceedings of the 9th Australian Computing Science Conference, January, 1986 - see PPRR-21-85.

Internal Reports

Morrison, R.
"S-Algol language reference manual", University of St Andrews CS-79-1, 1979.

Bailey, P.J., Maritz, P. & Morrison, R.
"The S-algol abstract machine", University of St Andrews CS-80-2, 1980.

Atkinson, M.P., Hepp, P.E., Ivanov, H., McDuff, A., Proctor, R. & Wilson, A.G.
"EDQUSE reference manual", Department of Computer Science, University of Edinburgh, September 1981.

Hepp, P.E. and Norrie, M.C.
"RAQUEL: User Manual", Department of Computer Science Report CSR-188-85, University of Edinburgh.

Norrie, M.C.
"The Edinburgh Node of the Proteus Distributed Database System", Department of Computer Science Report CSR-191-85, University of Edinburgh.

In Preparation

Kulkarni, K.G. & Atkinson, M.P.
"EFDM : Extended Functional Data Model", to be published in The Computer Journal.

Kulkarni, K.G. & Atkinson, M.P.
"EFDM : A DBMS based on the functional data model", to be submitted.

Atkinson, M.P. & Buneman, O.P.
"Database programming languages design", submitted to ACM Computing Surveys - see PPRR-17-85.

Morrison, R., Dearle, A., Bailey, P., Brown, A. & Atkinson, M.P.
"An integrated graphics programming system", to be presented at EUROGRAPHICS UK, Glasgow University, March 1986 - see PPRR-14-86.

Theses

The following Ph.D. theses have been produced by member of the group and are available from

The Secretary,
Persistent Programming Group,
University of Glasgow,
Department of Computing Science,
Glasgow G12 8QQ,
Scotland.

W.P. Cockshott
Orthogonal Persistent, University of Edinburgh, February 1983.

K.G. Kulkarni
Evaluation of Functional Data Models for Database Design and Use, University of Edinburgh, 1983.

P.E. Hepp
A DBS Architecture Supporting Coexisting Query Languages and Data Models, University of Edinburgh, 1983.

G.D.M. Ross
Virtual Files: A Framework for Experimental Design, University of Edinburgh, 1983.

G.O. Owoso
Data Description and Manipulation in Persistent Programming Languages, University of Edinburgh, 1984.

Persistent Programming Research Reports

This series was started in May 1983. The following list gives those produced and those planned plus their status at 17th March 1986.

Copies of documents in this list may be obtained by writing to
The Secretary,
The Persistent Programming Research Group,
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ.

PPRR-1-83	The Persistent Object Management System - Atkinson, M.P., Chisholm, K.J. and Cockshott, W.P.	[Printed]	£1.00
PPRR-2-83	PS-algol Papers: a collection of related papers on PS-algol - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R.	[Printed]	£2.00
PPRR-3-83	The PS-algol implementor's guide	[Withdrawn]	
PPRR-4-83	The PS-algol reference manual - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R.	[Printed]	£2.00
PPRR-5-83	Experimenting with the Functional Data Model - Atkinson, M.P. and Kulkarni, K.G.	[Printed]	£1.00
PPRR-6-83	A DBS Architecture supporting coexisting user interfaces: Description and Examples - Hepp, P.E.	[Printed]	£1.00
PPRR-7-83	EFDM - User Manual - K.G.Kulkarni	[Printed]	£1.00
PPRR-8-84	Progress with Persistent Programming - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R.	[Printed]	£2.00
PPRR-9-84	Procedures as Persistent Data Objects - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R.	[Printed]	£1.00
PPRR-10-84	A Persistent Graphics Facility for the ICL PERQ - Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. and Dearle, A.	[Printed]	£1.00
PPRR-11-85	PS-algol Abstract Machine Manual	[Printed]	£1.00
PPRR-12-85	PS-algol Reference Manual - second edition	[Printed]	£2.00
PPRR-13-85	CPOMS - A Revised Version of The Persistent Object Management System in C - Brown, A.L. and Cockshott, W.P.	[Printed]	£2.00
PPRR-14-86	An Integrated Graphics Programming Environment - second edition - Morrison, R., Brown, A.L., Dearle, A. and Atkinson, M.P.	[Printed]	£1.00
PPRR-15-85	The Persistent Store as an Enabling Technology for Integrated Project Support Environment - Morrison, R., Dearle, A, Bailey, P.J., Brown, A.L. and Atkinson, M.P.	[Printed]	£1.00
PPRR-16-85	Proceedings of the Persistence and Data Types Workshop, Appin, August 1985 - ed. Atkinson, M.P., Buneman, O.P. and Morrison, R.	[Printed]	£15.00
PPRR-17-85	Database Programming Language Design - Atkinson, M.P. and Buneman, O.P.	[Printed]	£3.00
PPRR-18-85	The Persistent Store Machine - Cockshott, W.P.	[Printed]	£2.00
PPRR-19-85	Integrated Persistent Programming Systems - Atkinson, M.P. and Morrison, R.	[Printed]	£1.00
PPRR-20-85	Building a Microcomputer with Associative Virtual Memory - Cockshott, W.P.	[Printed]	£1.00
PPRR-21-85	A Persistent Information Space Architecture - Atkinson, M.P., Morrison, R. and Pratten, G.D.	[Printed]	£1.00
PPRR-22-86	Some Applications Programmed in a Persistent Language - Cooper, R.L., Cranston, R.D., Dearle, A. and MacFarlane, D.K.	[In Preparation]	
PPRR-23-86	PS-algol Applications Programming - Cooper, R.L., Dearle, A., MacFarlane, D.K. and Philbrow, P.	[In Preparation]	
PPRR-24-86	A Compilation Technique for a Block Retention Language - Cockshott, W.P.and Davie, A.J.T.	[In Preparation]	
PPRR-25-86	Thoughts on Concurrency - Wai, F.	[In Preparation]	

PPRR-26-86 An Exception Handling Model in a Persistent Programming Language - Philbrow, P. [In Preparation]

PPRR-27-86 Concurrency in Persistent Programming Languages - Krablin, G.K. [In Preparation]

PPRR-28-86 A Domain Theoretic Approach to Higher-Order Relations - Buneman, O.P. [Printed] £1.00

PPRR-29-86 Extracting Garbage and Statistics from a Persistent Store - Campin, J. [In Preparation]

PPRR-30-86 Data Types for Data Base Programming - Buneman, O.P. [Printed] £1.00