An Introduction to the Formal Specification of Relational Query Languages

R. TURNER AND B. G. T. LOWDEN*
Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, Essex, UK

An introduction is given to the use of formal semantics as a means of both specifying relational query languages and of establishing a conceptual basis for their analysis. The approach is demonstrated with respect to the relational calculus and then extended to cope with the more advanced constructs of SEQUEL and QUEL.

1. INTRODUCTION

Following the publication of Codd's early work on the organisation of databases, the advantages of the relational approach have been increasingly well recognised and accordingly become the subject of many publications over the last few years.

One important area which has attracted much attention is the design, implementation and optimisation of relational query languages, either to be used in conjunction with a conventional host language or as stand-alone systems aimed at the casual user.

Early query languages were, to a large extent, based on either the relational algebra or relational calculus, both of which were first proposed by E. F. Codd.[1] The algebra employs a set of high-level procedural operators such as projection and join, whereas the calculus is a non-procedural, or declarative, language involving the use of free and bound variables and mathematical quantifiers. The concise notation of these languages assumes a certain degree of mathematical sophistication on the part of the programmer, and in recent years a number of other query languages have been developed, more appropriate to the non-technical user.[2]

As these languages have become more advanced, so the underlying semantic constructs have become more complex. However, with few exceptions,[3,4] language descriptions presented in the literature are based on a series of explanatory examples and rely heavily on the intuition of the reader in interpreting the text. We believe this state of affairs to be very unsatisfactory, and contend that the arguments for this belief are not dissimilar to those which led to the development of semantic techniques for the specification and analysis of conventional programming languages.

Early definitions of Algol 60, for example, consisted of a formal syntax together with an English language description of its semantic constructs. Being open to interpretation this meant that what constituted the actual language was defined by the way in which it was implemented. This resulted in a variety of different dialects rather than a single standard Algol 60.

These problems were addressed through the development of rigorous formalisms for specifying the semantics of a language. Each individual construct may be defined within a mathematical framework which is precise and unambiguous, and the resulting specification is sufficiently abstract to be independent of detailed implementation decisions.

In general query languages are not as large as programming languages, although many of them tend to exhibit features which in some ways make them more complex. The English-like appearance of SEQUEL,[5,6] for example, can make it more subject to error than conventional languages, since the way in which different parts of the syntax interact may not be immediately obvious from the language definition. This problem is often compounded by the fact that many query languages are presented, in the literature, by an informal syntax together with a few selected examples of their use. Often these examples do not highlight the real problem areas of the language.

Another issue is the way in which semantically ill-formed queries are interpreted. Let R be a relation of the form

$$\{\text{NAME, MANAGER, SALARY}\}.$$

Consider now a query requesting the names of all employees whose salary is greater than 'Smith'. Syntactically this is acceptable in most query languages although it clearly yields a nonsense result. It is simply not possible to compare numbers with names (strings on some alphabet). Exactly what happens with such a query is seldom, if ever, discussed in the literature and even then it is in the context of a specific implementation decision rather than a language design feature. Note that this problem is not the same as generating a well-formed but incorrect query[7] or a well-formed query which is then processed against a database containing null-values.[8]

In this paper we provide a formal semantic framework which can be employed in tackling these and other problems of relational query language analysis and, at the same time, provide a deeper understanding of their underlying constructs.

The semantic techniques are demonstrated with respect to a form of the relational calculus and also to two well-known query languages SEQUEL (now SQL) and QUEL.[9,10,11] For each language a set of recursively defined functions is provided which map syntactically correct constructs on to elements of certain sets. These domains or sets form a second component of any such formal specification. Our objective has been no more than to demonstrate the viability of the approach; however, in providing these definitions we believe that we have developed a framework in which any relational query language can be formally defined.

2. RELATIONAL CALCULUS

Our objective in this section is to illustrate the ideas and techniques involved in providing a denotational definition

* To whom correspondence should be addressed.

of a relational query language. We choose to study a variant of the relational calculus, due to Ullman, largely because it is syntactically rather simple and therefore facilitates an exposition of the main ideas without excessive syntactic complexity. All of what we shall say about the semantics of the calculus is implicit in Ullman's discussion; the formal semantics will merely make explicit and precise our underlying semantic intuitions. This section is, therefore, largely pedagogical; its purpose is to illustrate the main ideas behind the formal semantics of query languages.

Before we begin our discussion of the calculus itself we need to decide upon some notation for relations. As a first step we assume there is given some collection of disjoint domains or sets $\{D_s : s \in S\}$ of basic objects. This collection will include, for example, a domain of integers, a domain of strings (on some alphabet) and a domain of characters. This collection will remain fixed for the rest of the paper. Very roughly, relations will be represented as certain finite subsets of tuples, where tuples are finite sequences of elements from the domains D_s. To be more precise about this we need to be more precise about the tuples themselves. To facilitate our exposition we let $W = S^*$ be the set of finite sequences of elements from S, our index set for domains. We shall employ the notation $w = w(1), \ldots, w(n)$ where $w \in W$ is a sequence of length n whose ith component is $w(i)$, for $1 \leqslant i \leqslant n$. A tuple t is then just an element of some D^w where

$$D^w = D_{w(1)} x --- x D_{w(n)}$$

and the domain of tuples is given as

$$T = \bigcup_{w \in W} D^w.$$

For $t \in D^w$, with length n, the ith component of t is the component in $D_{w(i)}$ for $1 \leqslant i \leqslant n$.

Relations can then be identified with those finite subsets of tuples R, such that $R \leqslant D^w$ for some $w \in W$. In other words, the domain REL of relations is given as

$$REL = \{R : \exists\ w \in W \quad \text{such that} \quad R \in P_F(D^w)\}$$

where for any set X, $P_F(X)$ is the set of *finite* subsets of X.

Expressions in the relational calculus are of the form

$$\{x : f(x)\},$$

where x is a tuple variable and f is some formulae built up from atoms and logical operators. More precisely, the abstract syntax of the calculus is given as follows.

The domain section specifies what single-letter symbols are to be used as metavariables and over what domains they are to range.

2.1 Syntax of the calculus

2.1.1 *Syntactic domains*

$x \in$ vars	variables
$r \in$ rn	relation names
$o \in$ op	dyadic operator names
$n \in$ nml	numerals
$a \in$ atom	atomic well-formed formulae
$f \in$ wff	well-formed formulae
$e \in$ exp	relational calculus expressions

2.1.2 *Syntactic clauses*

$e ::= \{x : f(x)\} - x$ is the only free variable of f

$a ::= x_1[\bar{n}]\, o\, x_2[\bar{m}] \mid r(x) \mid \bar{n} o x[\bar{m}] \mid x[\bar{m}]\, o \bar{n}$

$f ::= a \mid f_1 \wedge f_2 \mid \sim f_1 \mid \exists x f_1$

The idea behind formal semantics is simple enough: each expression or term in the language is to be assigned an element of some set (domain) where the term set is to be understood in its mathematical sense. So, for example, well-formed formulae in our calculus will (ultimately) denote elements of the domain BOOL of truth-values, whereas expressions will denote elements in the domain P of expressible values.

Before we can provide the semantics of the calculus, however, we must deal with the problem of free variables in calculus well-formed formulae (roughly, a variable x is 'free' if it is not governed by a quantifier, see Ullman[3] for an exact definition). We must provide some way of binding the values of free variables to tuple values. To deal with this problem we introduce the notion of a local environment which is a function p: vars $->$ T. Our next semantic domain then, is the domain LENV of local environments

$$p \in \text{LENV} = [\text{vars} -> T],$$

which is the domain of functions from variables to tuples.

We now collect together all the semantic domains necessary to define the semantics of the calculus. With each domain we shall indicate what variable we shall use to range over its elements.

2.1.3 *Semantic domains*

$t \in T = \bigcup_{w \in W} D^w$	tuples
$R \in \text{REL} = \{R : \exists\ w \in W \text{ such that } R \in P_F(D^w)\}$	relations
$n \in N$	integers
$b \in \text{BOOL} = \{\text{true, false}\}$	boolean values
$p \in \text{LENV} = \{\text{vars} -> T\}$	local environments
$X \in P = P(T)$	the power set of T; the domain of expressible values

The last domain is the only one we have not met so far. It is the domain whose elements (sets of tuples) are the result of evaluating calculus expressions of the form $\{x : f(x)\}$.

We are now in a position to define the semantic functions for the calculus. There are just two:

V: wff $->$ [LENV $->$ BOOL $\cup$ {ERROR}]

E: exp $->$ P $\cup$ {ERROR}.

These functions will be defined by a recursive definition below. First, however, we need to say a little about the meaning of 'ERROR'. Certain expressions, and wff, in the calculus, although syntactically well formed, can produce an error. For example, suppose we attempt to select the ith component of a tuple which has no ith component. The value 'ERROR' is included to formally allow for this possibility. We shall have more to say about this later, but first we proceed to the formal definitions of V and E. We shall enclose all syntactic components of the definition (i.e. expressions and wff of the calculus) between brackets [ ].

$$(V1)\quad V[x_1[\bar{n}]ox_2[\bar{m}]]p = \begin{cases} \text{true} - \text{if } p(x_1)\downarrow n \neq \text{ERROR and} \\ \quad p(x_2)\downarrow m \neq \text{ERROR and} \\ \quad p(x_1)\downarrow n \varepsilon D_o \text{ and } p(x_2)\downarrow m \in D_o \\ \quad \text{and } p(x_1)\downarrow n\,\hat{o}\,p(x_2)\downarrow m \\ \text{false} - \text{if } p(x_1)\downarrow n \neq \text{ERROR and} \\ \quad p(x_2)\downarrow m \neq \text{ERROR and} \\ \quad p(x_1)\downarrow n \in D_o \text{ and } p(x_2)\downarrow m \in D_o \\ \quad \text{and it is false that} \\ \quad p(x_1)\downarrow n\,\hat{o}\,p(x_2)\downarrow m \\ \text{ERROR} - \text{otherwise} \end{cases}$$

where $\hat{o}$ is the comparison operator named by o (and $\hat{o}$ is an operation on $D_o x D_o$) and where

$$t\downarrow n \begin{cases} \text{the } n\text{th component of } t - \text{if there is one} \\ \text{ERROR} \qquad\qquad - \text{otherwise} \end{cases}$$

We insist that $p(x_1)\downarrow n \neq$ ERROR and $p(x_2)\downarrow m \neq$ ERROR, to ensure that we have objects to compare; we insist that they are both in D_o to ensure that they are comparable with respect to $\hat{o}$.

For example, $\hat{o}$ might be $\leqslant$ on the integers so that $D_o = N$.

$$(V2)\quad V[r(x)]p = \begin{cases} \text{true} - \text{if } p(x)\in D^w \text{ where } R \leqslant D^w \text{ and} \\ \quad p(x) \in R \\ \text{false} - \text{if } p(x)\in D^w \text{ where } R \leqslant D^w \text{ and it is} \\ \quad \text{false that } p(x)\in R \\ \text{ERROR} - \text{otherwise} \end{cases}$$

where r names R.

For a non-error result we need to check that $p(x)$ is an appropriate tuple to be presented to R. Formally this amounts to checking that $p(x)\in D^w$ where $R \leqslant D^w$.

$$(V3)\quad V[\bar{n}ox[\bar{m}]]p = \begin{cases} \text{true} - \text{if } p(x)\downarrow m \neq \text{ERROR and} \\ \quad p(x)\downarrow m\in N \text{ and } n\hat{o}p(x)\downarrow m \\ \text{false} - \text{if } p(x)\downarrow m \neq \text{ERROR and} \\ \quad p(x)\downarrow m\in N \text{ and it is false that} \\ \quad n\hat{o}p(x)\downarrow m \\ \text{ERROR} - \text{otherwise} \end{cases}$$

$$(V4)\quad V[x[\bar{m}]o\bar{n}]p = \begin{cases} \text{true} - \text{if } p(x)\downarrow m \neq \text{ERROR and} \\ \quad p(x)\downarrow m\in N \text{ and it is false that} \\ \quad p(x)\downarrow m\hat{o}n \\ \text{false} - \text{if } p(x)\downarrow m \neq \text{ERROR and} \\ \quad p(x)\downarrow m\in N \text{ and it is false that} \\ \quad p(x)\downarrow m\hat{o}n \\ \text{ERROR} - \text{otherwise} \end{cases}$$

In both ($V3$) and ($V4$) we need to insist that $p(x)\downarrow m\in N$ – otherwise no comparison can be made

$$(V5)\quad V[f_1\hat{\ }f_2]p = \begin{cases} \text{true} - \text{if } V[f_1]p = \text{true and} \\ \quad V[f_2]p = \text{true} \\ \text{false} - \text{if } V[f_1]p = \text{false or} \\ \quad V[f_1]p = \text{false} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

$$(V6)\quad V[\sim f]p = \begin{cases} \text{true} - \text{if } V[f]p = \text{false} \\ \text{false} - \text{if } V[f]p = \text{true} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

Observe that we have truth–value gaps: errors generated by atoms are inherited by more complex wff's of which they are part.

$$(V7)\quad V[\exists xf]p = \begin{cases} \text{true} - \text{if for } \textit{some} \\ \quad t\in T, V[f]p[t/x] = \text{true} \\ \text{false} - \text{if for } \textit{each} \\ \quad t\in T\ V[f]p[t/x] = \text{false} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

where $p[t/x]$ is that environment function identical to p except that it assigns the value t to x.

This brings us to the calculus expressions themselves:

$$(E)\quad E[\{x{:}f(x)\}]p = \begin{cases} \{t\in T{:}\ V[f(x)]p[t/x] = \text{true}\} - \text{if} \\ \quad \{x{:}f(x)\} \text{ is `sensible'} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

Here 'sensible' means that we need to ensure that there is some $t\in T$ such that

$$V[f(x)]p[t/x] = \text{ERROR}.$$

This is because we actually only want to consider those tuples for which it makes sense to ask whether or not they satisfy $f(x)$. For example, consider the (syntactically) well-formed query

$$\{x{:}r(x)\,x[\bar{1}] > \overline{27}\},$$

where R, the relation named by r, is a relation of the form ⟨NAME, MANAGER, SALARY⟩. It would be at best misleading to say there are no employers who satisfy $r(x)\hat{\ }x[\bar{1}] > \overline{27}$. The query is just 'ill formed', and since the syntax does not recognize this fact the semantics must.

This completes our discussion of the calculus. We now apply and extend the techniques developed here to a more 'English-like' query language.

3. THE FORMAL SEMANTICS OF SEQUEL

SEQUEL was designed as a database sublanguage for both the professional programmer and the casual user and is currently used as a basis for data manipulation in system R, a relational database system under development at IBM San José.

The fundamental command/query of the SEQUEL language is called a 'mapping' and has the form:

SELECT ATTRIBUTE
FROM Table name/Relation name
WHERE Boolean condition

For example, the query

(*a*) *SELECT* name
FROM EMP
WHERE EMP ↓ DNO = 50

will select the names of employees in DEPT 50 where the relation EMP has the following form:

```
EMP:  EMP NO  NAME  DNO  JOB  MGR  SAL  COMM
```

To begin with, we introduce the syntax of SEQUEL. For pedagogical reasons we shall only study a subset/variant of SEQUEL(SS) but which nevertheless illustrates all the important semantic issues.

3.1 Syntactic domains

$q\in$ query	queries
b ∈ bexp	Boolean expressions
e ∈ exp	expressions
r ∈ rn	relation names
o ∈ op	operation names
O ∈ so	set-operation names
c ∈ cons	constant symbols
n̄ ∈ nml	numerals

Note that numerals will be distinguished from numbers by the use of the bar notation, e.g. n̄ is the numeral denoting the number n.

3.2 Syntactic clauses

$q ::= \mathit{SELECT}\ \bar{n}\ \mathit{FROM}\ r\ \mathit{WHERE}\ b$

$b :: \quad e_1 o e_2 \mid q_1 O\ q_2 \mid e\ \mathit{IN}\ q$

$e :: \quad c \mid \bar{n} \mid r \downarrow \bar{n}$

Some comments regarding the difference between SS and SEQUEL are necessary. Perhaps the most obvious difference concerns the attribute names. We have, for simplicity, assumed that they are coded as integers. The syntax is not explicit about the operations and set-operations but the intention is that op includes a name for $\leqslant$, on the integers, whereas a typical set-operation would be the inclusion relation on sets. The syntax of SS is much simpler than that of SEQUEL but we have tried to isolate the important semantic issues without introducing undue syntactic complexity. The reader should be able to provide a semantics for the whole of SEQUEL once the section has been mastered.

Before we can develop the semantics of SS we need to investigate the problem of how to interpret the occurrence of table means in *WHERE* expressions. Consider the expression

$$\mathrm{EMP} \downarrow \mathrm{DNO} = 50$$

from our initial example. The reference to EMP here (actually implicit in SEQUEL) is not the whole relation EMP but to a specific member. To model this we once more introduce a local environment

$$p \in \mathrm{LENV} = [\mathrm{rn} \to \mathrm{T}],$$

which as we shall see forces the binding of the occurrence of relation names inside such expressions to specific tuples.

As in the case of the calculus, our first task has to be the presentation of the semantic domains of SS.

3.3 Semantic domains of SS

$t \in T = \bigcup_{w \in W} D^w$	tuples
$R \in \mathrm{REL} = \{R : \exists w \in W \text{ such that } R \in P_F(D^w)\}$	relations
$n \in N$	integers
$b \in \mathrm{BOOL} = \{\text{true}, \text{false}\}$	Boolean values
$p \in \mathrm{LENV} = [\mathrm{rn} \to T]$	local environments
$X \in P = \{X : \exists s \in S \text{ such that } X \in P_F(D_S)\}$	query values
$v \in V = \bigcup_{s \in S} D_S$	values

These domains are much the same as those for the calculus. There are, however, a few differences worth mentioning. The domain LENV is essentially the same, but since there are no explicit range statements in SS (or SEQUEL for that matter) relation names play the role of tuple variables. The domain V does not occur in our account of the calculus. In SS, however, expressions evaluate to actual values and queries to finite sets of values rather than to finite sets of tuples.

We can now proceed to the formal specification of SS. The definition of SS involves the stipulation of three (mutually) *recursive* functions Q, B E which have functionality

$Q : \text{query} -> [\mathrm{LENV} -> P \cup \{\text{Error}\}]$

$B : \text{bexp} -> [\mathrm{LENV} -> \text{Bool} \cup \{\text{Error}\}]$

$E : \text{exp} -> [\mathrm{LENV} -> V \cup \{\text{Error}\}]$

The first evaluates the queries themselves while the second and third return the values of Boolean expressions and expressions respectively. The ERROR value is included for the same reason as in the calculus – the syntax does not prevent the formation of semantically ill-formed expressions and queries. These three functions are defined by simultaneous or mutual recursion as follows.

(Q.) $Q[\mathit{SELECT}\ \bar{n}\ \mathit{FROM}\ r\ \mathit{WHERE}\ b]p =$
$\{t \downarrow n : t \in R \text{ and } B[b]p[t/r] = \text{true}\}$ – if for some $t \in R$, $B[b]p[t/r] \neq$ ERROR and R has at least n-components
ERROR – otherwise

where r names the relation R.

The function Q only returns a value when the Boolean condition, applied to some/each tuple in the relation R, returns true or false and when R has at least n components, i.e. the length of tuples in R is at least n; otherwise the query results in an error. The stipulation that $B[b]\,[t/r] \neq$ ERROR, for some $t \in R$, is to ensure that R is an 'appropriate' relation to occur in b. For example, consider a relation R of the form ⟨NAME, MANAGER, SALARY⟩ and the Boolean expression b, $r \downarrow \bar{2} > \overline{27}$. This is syntactically well formed, but semantically it is nonsense. If we did not include the stipulation that $B[b]p[t/r] \neq$ ERROR, for some tuple in R, the query '$\mathit{SELECT}\ \bar{1}\ \mathit{FROM}\ r\ \mathit{WHERE}\ b$' would return the empty set of names – a misleading answer at best.

Next we provide the definition of E.

(E1) $\quad$ for all $\bar{n} \in \text{nml}$, $E[\bar{n}]p = n$

(E2) $\quad$ for all $c \in \text{cons}$, $E[c]p = \hat{c}$

Hence E2 shows that the value $\hat{c}$ of the constant symbol c is independent of the environment.

(E3) $\quad E[r \downarrow n]p = p(r) \downarrow n$

Here we select the nth component of p(r).

Observe that one reason for the inclusion of the ERROR value occurs in (E3) and proliferates throughout the definition of SS. If we try to select the nth component of a tuple which does not have one we should expect to get an ERROR result.

Finally, we provide the specification of B, the semantic function for boolean expressions.

$$\text{(B1)}\quad B[e_1 o e_2]p = \begin{cases} \text{true – if } E[e_1]p \neq \text{ERROR and} \\ \quad E[e_2]p \neq \text{ERROR and} \\ \quad E[e_1]p \in D_o \text{ and } E[e]p \in D_o \text{ and} \\ \quad E[e_1]p\ \hat{o}\ E[e_2]p \\ \text{false – if } E[e_1]p \neq \text{ERROR} \\ \quad \text{and } E[e_2]p \neq \text{ERROR and} \\ \quad E[e_1]p \in D_o \text{ and } E[e_2]p \in D_o \text{ and} \\ \quad \text{it is false that } E[e_1]p\ \hat{o}\ E[e_2]p \\ \text{ERROR otherwise} \end{cases}$$

where $\hat{o}$ is the operator named by o and has range $D_o \times d_o$. We shall employ the $\hat{o}$ notation throughout to distinguish the syntactic operator o from the corresponding relation $\hat{o}$.

Once again we have to check that $E[e_1]p$ and $E[e_2]p$ return proper values ($\neq$ ERROR) and are suitable for comparison; unfortunately, the syntax does not perform this task for us.

$$\text{(B2)}\ B[q_1 O q_2]p = \begin{cases} \text{true – if } Q[q_1]p \neq \text{ERROR and } Q[q_2]p \neq \text{ERROR and are contained in some } D_s \text{ and } Q[q_1]p\,\hat{O}\,Q[q_2]p \\ \text{false – if } Q[q_1]p \neq \text{ERROR and } Q[q_2]p \neq \text{ERROR and are contained in some } D_s \text{ and not } Q[q_1,]p\,\hat{O}\,Q[q_2]p \\ \text{ERROR – otherwise} \end{cases}$$

where $\hat{O}$ is the set-operator named by O.

We have to check that both queries do not evaluate to ERROR. The set-operator $\hat{O}$ might be, for example, $\leqslant$ and $Q[q_1]p\,\hat{O}\,Q[q_2]p$ would hold just in case $Q[q_1]p \leqslant Q[q_2]p$.

$$\text{(B3)}\ B\{e\ \mathit{in}\ q]p = \begin{cases} \text{true – if } E[e]p \neq \text{ERROR, and } Q[q_1]p \neq \text{ERROR, and } E[e]p \in D_s \text{ where } Q[q]p \leqslant D_s, \text{ and } E[e]p \in Q[q]p \\ \text{false – if } E[e]p \neq \text{ERROR, and } Q[q]p \neq \text{ERROR, and } E[e]p \in D_s \text{ where } Q[q]p \leqslant D_s, \text{ and } E[e]p \in Q[q]p \\ \text{ERROR – otherwise} \end{cases}$$

Here we need to check that E[e]p ≠ ERROR, Q[Q]p ≠ ERROR; and that E[e]p is the same type of object as those in Q[Q]p, for the result not to be ERROR.

This completes our discussion of the formal semantics. We now illustrate the semantics by way of an example.

4. EXAMPLE

Find the names of employees who work for departments in Evanston.

(*b*) *SELECT* Name
FROM EMP
WHERE EMP↓DNO *IN*
SELECT DNO
FROM DEPT
WHERE DEPT↓LOC = EVANSTON

(Observe that, in SS, attributes are coded as numerals; for convenience we shall use the attributes themselves).

We first apply Q to (b) to obtain

$\{t\downarrow \text{name} : t \in \widehat{\text{EMP}}$ and
B[EMP↓DNO *IN SELECT* DNO
FROM DEPT
WHERE DEPT↓LOC = EVANSTON
$]p\,[t/\widehat{\text{EMP}}] = \text{true}$
}.

We now unwrap the application of B using (B3) to obtain

B[EMP↓DNO *IN SELECT* DNO
FROM DEPT

WHERE DEPT↓LOC = EVANSTON]$p[t/\widehat{\text{EMP}}]$ = true

if and only if (using (B3))

E[EMP↓DNO] ≠ ERROR and Q[*SELECT* DNO
FROM DEPT
WHERE DEPT↓LOC = EVANSTON] ≠ ERROR

and

E[EMP↓DNO]$p[t/\widehat{\text{EMP}}] \in D_s$ where
Q[*SELECT* DNO
FROM DEPT
WHERE DEPT↓LOC = EVANSTON]$p[t/\widehat{\text{EMP}}] \leqslant D_s$

and

E[EMP↓DNO]$p[t/\widehat{\text{EMP}}] \in$ Q *SELECT* DNO
FROM DEPT
WHERE DEPT↓LOC = EVANSTON]$p[t/\widehat{\text{EMP}}]$

By using (E3) E(EMP↓DNO]$p[t/\widehat{\text{EMP}}]$ may be simplified to

$$p[t/\widehat{\text{EMP}}](\text{EMP})\downarrow \text{DNO} = t\downarrow \text{DNO}$$

and using (Q) the final Q-expression above becomes

$$\{t'\downarrow \text{DNO} : t' \in \widehat{\text{DEPT}} \text{ and } B[\text{DEPT}\downarrow\text{LOC} = \text{EVANSTON}]$$
$$p[t/\text{EMP}][t'/\widehat{\text{DEPT}}] = \text{true}\}$$

Using (B1), (E2) the above B clauses simplifies to

$$t'\downarrow\text{LOC} \neq \text{ERROR and EVANSTON} \neq \text{ERROR and } t'\downarrow\text{LOC} = \text{EVANSTON}$$

Hence the Q clause may be further simplified to

$$\{t'\downarrow\text{DNO} : t' \in \widehat{\text{DEPT}} \text{ and } t\downarrow\text{LOC} \neq \text{ERROR and EVANSTON} \neq \text{ERROR}$$
$$\text{and } t'\downarrow\text{LOC} = \text{EVANSTON}\}$$

Finally, the application of Q to (b) may be written as

$$\{t\downarrow\text{name} : t \in \widehat{\text{EMP}} \text{ and } t\downarrow\text{DNO} \in \{t'\downarrow\text{DNO} : t' \in \widehat{\text{DEPT}} \text{ and } t'\downarrow\text{LOC} = \text{EVANSTON}\}\}$$

together with the relevant error checks.

5. THE FORMAL SEMANTICS OF QUEL

We now turn to a more elaborate Query language, namely QUEL. QUEL is the query language component of INGRES (Interactive Graphics and Retrieval System). It is a calculus-based language which is closely modelled on the language ALPHA. There is, however, one significant difference: QUEL is free of all explicit quantifiers.

Each query of QUEL contains one or more 'range' statements and one or more retrieve statements. The basic form of a QUEL query is

RANGE STATEMENT(S)
RETRIEVE INTO V
RESULT DOMAIN = function
.
.
.
RESULT DOMAIN = function
WHERE QUALIFICATION

We provide the precise syntax of QUEL in Fig. 3. We have simplified QUEL by (for example) only allowing the retrieval into one named relation, but this does not suppress any important semantic issues.

Our first objective must be to 'lay out' the semantic domains.

5.1 Semantic domains of QUEL

$t \in T = \bigcup_{w \in W} D^w$ tuples
$R \in \mathrm{REL} = \{R : \exists w \in W \text{ such that } R \in P_F(D^w)\}$ relations
$v \in V = \bigcup_{s \in S} D_s$ values
$p \in \mathrm{LENV} = [\mathrm{var} -> T]$ local environments
$M \in \mathrm{GENV} = [\mathrm{var} -> \mathrm{REL}]$ global environments
$D \in DB = [nrn -> \mathrm{REL}]$ data base
$b \in \mathrm{Bool} = \{\text{true, false}\}$ boolean values
$X \in P = P_F(V)$ set values

5.2 Abstract syntax of QUEL

5.2.1 *Syntactic domains*

$i \in$ Query	queries
$f \in$ fun	function expressions
$q \in$ qual	qualifications
$K \in$ set	sets
$\bar{n} \in$ Nml	numerals
$x \in$ var	variables
$k \in$ bset	basic sets
$p \in$ *nrn*	new relation names
$d \in$ ran	range statements
$O \in$ set-op	set-operation names
$r \in$ rn	basic relation names
$C \in$ set-fn	set functions

5.2.2 *Syntactic clauses*

$f ::= \bar{n} \mid x.\bar{n} \mid C(K) \mid f_1 + f_2 \mid f_1 * f_2$
$q ::= f_1 = f_2 \mid q_1 \hat{} q_2 \mid q_1 \vee q_2 \mid K_1 O K_2 \mid \bar{n} \in K$
$K ::= k \mid$ SET(f *WHERE* q) $\mid$ SET(f_1 *BY* f_2 *WHERE* q) $\mid K_1 \cup K_2 \mid K_1 - K_2 \mid K_1 \cap K_2$
C ::= COUNT | AGG
i ::= d : *RETRIEVE INTO p*
    f_1;
    .
    .
    .
    f_k;
  WHERE q
d ::= *RANGE* x is r | RANGE x is p | d_1 ; d_2

In providing the semantics of QUEL we require not only a 'local environment' but also a 'global environment' as well as an explicit representation of the database. The latter is required because the basic form of a QUEL query permits the construction of new named relations. Global environments are necessary because of the introduction of range statements. For example, the statement RANGE x is r forces the binding of x, in any local environment, to be an element of the relation named by r.

To define QUEL we need to stipulate five semantic functions:

F : fun $->$ [LENV $->$ V $\cup$ {ERROR}]

S : set $->$ [LENV $->$ P $\cup$ {ERROR}]

B : qual $->$ [LENV $->$ BOOL $\cup$ {ERROR}]

G : range $->$ [GENV $->$ GENV]

Q : queries $->$ [DB $->$ DB]

Once again these are defined by simultaneous recursion: as we shall see, the function F involves S; S involves F and B; B involves F and S, and Q involves F, B and G.

To begin with then we define the function F:

(F1) $F[\bar{n}]p = n$

(F2) $F[x.\bar{n}]p = p(x) \downarrow n$

(F3) $F[C(K)]p = \hat{C}(S[K]p)$

where

$\widehat{\mathrm{COUNT}}(S) =$ the number of elements in S

$$\widehat{\mathrm{AGG}}(S) = \begin{cases} \text{the aggregate of S} - \text{if } S \leqslant N \\ \text{ERROR} \quad - \text{otherwise} \end{cases}$$

(F4)
$$F[f+g]p = \begin{cases} F[f]p + F[g]p - \text{if } F[f]p \in N \text{ and } F[g]p \in N \\ \text{ERROR} - \text{otherwise} \end{cases}$$
(similarly for * etc.)

This is mostly straightforward. The function F either produces a value (in V) or an ERROR. The only complication occurs in (F3) where reference is made to the function S. So in order to grasp what is involved, we need to consider S itself.

(S1) $S[k]p = \hat{k}$

where $\hat{k}$ is some fixed element of $P_F(V)$ denoted by k.

(S2) S[SET(f *WHERE* q)]p =
  $\{F[f]p' \neq \text{ERROR} : B[q]p' = \text{true}\}$ if Z
  ERROR – otherwise

where Z is the statement that there is some p′ such that $B[q]p' \neq$ ERROR and $F[f]p' \neq$ ERROR.
We need to include the condition Z to ensure that the set-definition is 'sensible'. For example, suppose $f = \widehat{\mathrm{AGG}}(k)$ where k denotes some subset of strings and q is the Boolean expression $\bar{n} \in k$. Clearly there are no environments p for which $F[f]p \neq$ ERROR and $B[q]p \neq$ ERROR. Without this condition the result of SET(f *WHERE* q) would be the empty set, and this is intuitively unsound. We want to say the expression is semantically illformed, not that the set of values for f, where q holds, is empty. The condition ensures that there is a 'consistent' assignment to the variables of f and q.

(S3) S[SET(f BY g *WHERE* q)]p =
  $\{F[f]p' \neq \text{ERROR} : F[g]p = F[g]p' \neq \text{ERROR and } B[q]p' = \text{true}\}$ – if $\bar{Z}$
  ERROR – otherwise

where $\bar{Z}$ is the statement that for some p′, $F[f]p' \neq$ ERROR, $F[g]p = F[g]p' \neq$ ERROR and $B[q]p' \neq$ ERROR.

We shall later illustrate the use of this construct by way of example.

(S4) $S[K_1 \cup K_2]p = S[K_1]p \cup S[K_2]p$

where by stipulation S $\cup$ ERROR = ERROR $\cup$ S = ERROR, for any set S. Similar clauses apply for intersection and relative complement.

We next present the details of the semantic function for Boolean expressions or qualifications.

(B1) $B[f_1 = f_2]p =$

$$\begin{cases} \text{true} - \text{if } F[f_1]p \neq \text{ERROR and } F[f_2]p \neq \\ \quad \text{ERROR and are in some } D_S \text{ and are equal.} \\ \text{false} - \text{if } F[f_1]p \neq \text{ERROR and } F[f_2]p \neq \\ \quad \text{ERROR and are in some } D_S \text{ but are not equal.} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

(B2) $B[q_1 \wedge q_2]p =$
$$\begin{cases} \text{true} - \text{if } B[q_1]p = \text{true and } B[q_2]p = \text{true} \\ \text{false} - \text{if } B[q_1]p = \text{false or } B[q_2]p = \text{false} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

(B3) $B[K_1 \, O K_2]p =$
$$\begin{cases} \text{true} - \text{if } S[K_1]p \neq \text{ERROR and } S[K_2]p \neq \text{ERROR and are contained in some } D_S \text{ and satisfy } \hat{O} \\ \text{false} - \text{if } S[K_1]p \neq \text{ERROR and } S[K_2]p \neq \text{ERROR and are contained in some } D_S \text{ and do not satisfy } \hat{O} \\ \text{ERROR} - \text{otherwise} \end{cases}$$

where Ô is the set operator named by O.

$$\text{(B4)} \quad B[\bar{n} \in K]p = \begin{cases} \text{true} - \text{if } n \in \Sigma[K]\pi \leqslant N \\ \text{false} - \text{if } n \in S[K]p \leqslant N \\ \text{ERROR} - \text{otherwise} \end{cases}$$

We claimed that range statements create a global environment with respect to which all bindings of variables to tuples must be 'consistent'. We now present the details of the creation of such an environment.

$$G : \text{Range} -> [\text{DB} -> [\text{GENV} -> \text{GENV}]]$$

(G1) $G[\text{RANGE x is r}]DM = M[R \mid x]$

(G2) $G[\text{RANGE x is p}]DM = M[D(p) \mid x]$

(G3) $G[d_1 ; d_2]DM = G[d_2]DG[d_1]DM$

where, in (G1), M[R | x] is that global environment identical to M except it assigns the relation R to the variable x and where R is the relation named by r. Note the different roles played by basic relations and new named relations; the latter obtain their values from the database itself.

We can now provide the main semantic function for QUEL queries.

$$Q[d : \textit{RETRIEVE INTO } p \; f_1 ; f_2 ; \text{—} ; f_k \; \textit{WHERE } q]D$$
$$= \begin{cases} D[p \mid P] - \text{if there is an environment (local) p for which } \{G[d]D \perp\}Cp,\ F[f_i]p \neq \text{ERROR and } B[q]p \neq \text{ERROR for } 1 \leqslant i \leqslant k. \\ \text{ERROR} - \text{otherwise} \end{cases}$$

where $P = \{t \in T : t \in \{ < F[f_1]p \neq \text{ERROR}, \text{––}, F[f_n]p \neq \text{ERROR} \rangle : B[q]p = \text{true and } \{G[d]D \dagger\}Cp\}$
and $MCp \langle = \rangle (\forall x)(p(x) \in M(x))$
and $\perp \in \text{GENV}$ is given as $(\forall x)(\perp(x) = \emptyset)$ ($\emptyset$ the empty relation).

Then condition $\{G[d]D \perp\}Cp$ forces the bindings of all variables to be consistent with the range declarations. Observe that P is an element of REL, since a simple inductive argument shows that for some fixed function f the different values of F[f]p (as p varies) will be in the same domain.

We shall now, as promised, illustrate the use of the statement form SET(f *BY* g *WHERE* q) by way of an example.

6. EXAMPLE

Suppose 'supply' is a 3-place relation whose first place is occupied by supplier names, the second place by part numbers and the third by prices. Consider the query:

'Find the suppliers whose price for every part is greater than $10'

This can be represented in QUEL* by

```
(1) RANGE of x is SUPPLY
(2) RETRIEVE INTO p
        x. NAME
(3) WHERE COUNT (x. PARTNO BY x. NAME)  }
(4)        WHERE x. PRICE > $10)        }
(5)        =                            } q
(6)        COUNT (x. PARTNO BY x. NAME) }
```

(Note: 'x. PARTNO *BY* x. NAME' is short for 'x. PARTNO *BY* x. NAME *WHERE* NAME = NAME'.)
In what follows we shall omit the ERROR checks.

The range statement of line 1 forces the construction of a global environment M which binds the value of the variable x to the supplier relation. To evaluate the retrieve statement of line 2 we must compute all the values

(7) F[x. NAME]p

for those p which are compatible with M (i.e. MCp) and which satisfy the *WHERE* clause (see details of R). Now how do we compute those p which satisfy the *WHERE* clause? Let p be a candidate and let

(8) $v = F[\text{x. NAME}]p \neq \text{ERROR}$

We must check to see if B[q]p = true. This forces us to evaluate (by (B1))

(9) F[COUNT(x. PARTNO *BY* x. NAME *WHERE* x. PRICE > \$10)]p

and

(10) F[COUNT(x. PARTNO *BY* x. NAME)]p

We concentrate on (9). This reduces (by (F3) to

(11) $\widehat{\text{COUNT}}\{F[\text{x. PARTNO}]p' \neq \text{ERROR} : F[\text{x. NAME}]p' = F[\text{x. NAME}]p \neq \text{ERROR}$

and

$B[\text{x. PRICE}]p' > \$10\}$

The full import of (3) can now be seen. We only count those values F[x. PARTNO]p′ where F[x. NAME]p′ = v. In other words, we take the count of all the parts, which *v* supplies and which are greater than $10.

The second occurrence of the variable x in COUNT(x. PARTNO BY x. NAME *WHERE* x. PRICE > $10) plays a different role to the first and third occurrences and this is reflected by the role of the two environments p and p′. The range of values permitted by the first and third occurrences of x (the class of p′ permitted) is determined by the range of values permitted for the second occurrence (fixed by p). So the different conceptual roles played by these different instances of x are reflected by the constraining effect the choice of the first environment has on the selection of further environments. We believe that this is implicit in the original discussion of QUEL; the formal semantics merely makes this design feature precise and explicit.

7. CONCLUSIONS

In this paper we have given an introduction to the use of formal semantics as a means of rigorously defining and analysing the semantics of relational query languages.

* Note that we have not explicitly coded attributes as numerals but *x.* NAME, for example, would be rendered $x.\bar{1}$.

The approach adopted permits specification at a level of abstraction which is independent of any particular implementation. We have demonstrated this approach with respect to the calculus, QUEL, and SEQUEL.

Formal specifications provide a basis for communication and discussion between the language designer and the implementor which is also a concise and unambiguous reference standard for the user. At the design stage, the semantic analysis of proposed constructs can provide valuable insight into potential irregularities and undesirable interactions in the language and as such can play an important role in language development.

REFERENCES

1. E. F. Codd, *Relational Completeness of Data Base Sublanguages*. Data Base Systems, Courant Computer Science Symposia, vol. 6. Prentice-Hall, Englewood Cliffs, N.J. (May 1971).
2. BCS Query Language Group, *Query Languages – a Uniform Approach*. Heyden (1981).
3. J. D. Ullman, *Principles of Database Systems*. Pitman (1980).
4. D. Bjorner, Formalisation of database models. *Proceedings of the Winter School of Software Specification*, Copenhagen (1979).
5. M. H. Astrahan and D. D. Chamberlin, Implementation of a structured English query language. *CACM* 18, 10 (1975).
6. M. H. Astrahan, *A History and Evaluation of System R*. I.B.M. Research Report RJ2843 (1980).
7. R. Crowe et al., Query validation: reverse translation and the connection and selection trap. *Proceedings of the British National Conference on Databases* (1981).
8. C. J. Date, *Introduction to Database Systems*, vol. 2. *Addison Wesley* (1983).
9. G. D. Held, M. R. Stonebraker and E. Wong, INGRES – a relational database system. *Proceedings of the National Computer Conference*, no. 44 (1975).
10. M. R. Stonebraker et al., The design and implementation of INGRES. *ACM TODS* (1976).
11. D. D. Chamberlin, *A Summary of User Experience with the SQL Data Sublanguages* I.B.M. Research Report RJ2767 (1980).