NPS52-86-023

NAVAL POSTGRADUATE SCHOOL

Monterey, California

FOUR RELATIONAL PROGRAMS

B. J. MacLennan

November 1986

Approved for public release; distribution unlimited.

Prepared for:
Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

The work reported herein was supported by Contract from the Office of Naval Research.

This report was prepared by:

Chairman
Department of Computer Science

Dean of Information and
Policy Science

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-86-023	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FOUR RELATIONAL PROGRAMS		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bruce J. MacLennan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N0001485WR24057
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE November 1986
		13. NUMBER OF PAGES 24
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Relational programming, functional programming, relations, relational algebra, relational calculus, applicative programming, logic programming, combinator, very-high level language, Gaussian elimination, finite state automata, higher order functions

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Relational programming is a style of programming in which entire relations are manipulated as data and in which programs are also considered relations. Extensive use is made of higher-order, finite and infinite relations and functions.

In this report we demonstrate the relational programming language RPL by using it to develop four programs: (1) computing word frequencies from text; (2) minimizing deterministic finite state automata; (3) Gaussian elimination; and (4) updating an employee file. Transcripts of actual runs

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

are included in the appendices, as is a summary of the language.

FOUR RELATIONAL PROGRAMS[*]

B. J. MacLennan
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Abstract:

In this report we demonstrate the relational programming language RPL by using it to develop four programs. These programs are: (1) computing a table of word frequencies from a text; (2) minimizing a deterministic finite state automata; (3) Gaussian elimination; and (4) a simple data processing example involving updating an employee file. Appendix A shows transcripts of executions of the programs on the Brown and Mitton interpreter [Brown&Mitton]. The reader is presumed to be familiar with RPL, which is described in [MacLennan83]. For convenience, however, Appendix B contains the RPL grammar, and Appendix C describes the language accepted by the Brown and Mitton interpreter.

1. Computing Word Frequencies

The first example, which is adapted from [MacLennan83], it to compute a frequence table F from a text (sequence of words) S. That is, given S such that $S{\downarrow}i$ is the i^{th} word, we compute F such that $F{\downarrow}w$ is the frequency (number of occurrences) of word w in S. For an example, we take $S =$ <"to", "be", "or", "not", "to", "be">, which is just an abbreviation for the relation:

$$S = \{1\text{:``to''}, 2\text{:``be''}, 3\text{:``or''}, 4\text{:``not''}, 5\text{:``to''}, 6\text{:``be''}\}$$

Pictorially,

1	"to"
2	"be"
3	"or"
4	"not"
5	"to"
6	"be"

In this case, the desired frequency table F is:

$$F = \{\text{``to''}:2, \text{``be''}:2, \text{``or''}:1, \text{``not''}:1\}$$

Pictorially,

"to"	2
"be"	2
"or"	1
"not"	1

In other words, "to" occurs twice, "be" occurs twice, "or" occurs once, and "not" occurs once. Of course, since F is a relation, the order in which the elements are listed is irrelevant.

To develop the general word-counting program, we work through this particular example. Since the frequency table is a function *from* the words *to* their frequencies, the first step is to reverse columns of S:

$$S^{-1} = \{\text{``to''}:1, \text{``be''}:2, \text{``or''}:3, \text{``not''}:4, \text{``to''}:5, \text{``be''}:6\}$$

* The work reported herein was supported by Contract N00014-85-WR-24057 from the Office of Naval Research.

Pictorially,

"to"	1
"be"	2
"or"	3
"not"	4
"to"	5
"be"	6

Notice that this relation is not a function (i.e., it is not single valued). We can make it a function by forming the "unit image" of the table:[1]

$$\text{unimage } S^{-1} \;=\; \{\text{"to"}{:}\{1, 5\}, \text{"be"}{:}\{2, 6\}, \text{"or"}{:}\{3\}, \text{"not"}{:}\{4\}\}$$

Pictorially,

"to"	{1, 5}
"be"	{2, 6}
"or"	{3}
"not"	{4}

This tells us, for example, that the word "be" occurs in positions 2 and 6 in the text. We do not need to know the places where a given word occurs, but only the number of such places. Therefore, we send the preceding table through the **size** (cardinality) function (by the relative product operation):

$$\text{unimage } S^{-1} \mid \textbf{size} \;=\; \{\text{"to"}{:}2, \text{"be"}{:}2, \text{"or"}{:}1, \text{"not"}{:}1\}$$

This is the desired result; the final step is depicted in Figure 1. Notice that since **size** is defined for all sets, it is in effect an infinite relation; this is permitted in RPL.

The resulting program is:

$$F \equiv \text{unimage } S^{-1} \mid \textbf{size}$$

We can turn it into a function definition to compute the frequency table for any text S by:

$$\text{freq } S \;\equiv\; \text{unimage } S^{-1} \mid \textbf{size}$$

It remains to define the 'unimage' function, which is not built into RPL. On the other hand, RPL does have the builtin operator **unimg**, defined to that T **unimg** x is the set of all y such that $x{:}y \in T$. This can be used to define unimage. To see this, note that the *left section* [T **unimg**] is the function that takes any x into its image under T. Although T is finite (and extensional), sections are always intensional, so it is necessary to to convert [T **unimg**] to its extensional equivalent. This is accomplished with the RPL **restrict** operation, which converts an intensional relation to an extensional relation by restricting its domain to a finite set. Hence we define:

$$\text{unimage } T \;\equiv\; \textbf{dom } T \textbf{ restrict } [T \textbf{ unimg}]$$

The following is an example RPL session that defines the freq function an applies it to a particular text ('?>' is the RPL prompt):

1. This function is related to the RPL **unimg** operator; see below.

"to"	{1, 5}
"be"	{2, 6}
"or"	{3}
"not"	{4}

|

{}	0
{1}	1
{4}	1
{2, 6}	2
{4, 8}	2
{4, 6, 7}	3
⋮	⋮

||

"to"	2
"be"	2
"or"	1
"not"	1

Figure 1. Piping unimage S^{-1} Through **size** Function

?> unimage T $\equiv$ **dom** T **restrict** [T **unimg**]

?> freq S $\equiv$ unimage S^{-1} | **size**

?> freq <"to", "be", "or", "not", "to", "be">

{"be":2, "to":2, "not":1, "or":1}

?> **done**

The appendix contains the actual transcript of this RPL session; it shows how programs must be represented for the Brown and Mitton interpreter. The preceding formulas were produced from this transcript by a pretty printer.

2. Minimization of Deterministic Finite Automata

The next example program is the equivalence and minimization of deterministic finite automata by an algorithm developed by Robert Floyd[2]. We assume that we have a finite alphabet Σ and a finite set Q of states. The set $F \subseteq Q$ represents the *final* (accepting) states. The finite relation T is such that for $a \in \Sigma$, $T \downarrow a$ is the transition relation for the symbol a. That is, $<q, q'> \in T \downarrow a$ if and only if the symbol a takes state q into state q'.

2. Private communication, 1985.

Our goal is to define a relation R_∞ such that $<q, q'> \in R_\infty$ if and only if q and q' are *not* equivalent states. This is done in a series of steps, starting from pairs of states that are known to be inequivalent, namely the final and nonfinal states:

$$R_0 \equiv F \times (Q \setminus F)$$

We now work backward: any states that under the same input lead to inequivalent states are themselves considered inequivalent. For example, R_0 relates inequivalent states; R_1 relates states related by R_0 together with those that under the same input character are taken into states related by R_0; R_2 relates states related by R_0 together with those that under the same one or two input characters lead to states related by R_0; and so on. Each step of this process is accomplished by a function ψ; that is, we will define ψ so that $R_{i+1} = \psi R_i$. It will be easy to see that this process converges in $n = (\textbf{size } Q)^2$ steps, so

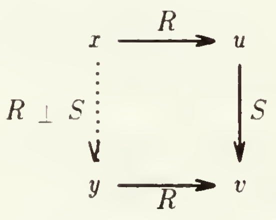
$$R_\infty \equiv \psi^n R_0$$

Next we consider ψ.

As a preliminary we define the *polymorphic image* of one relation under another relation. If R and S are two relations, then the polymorphic image under R of S, or more briefly the R-image of S, is defined

$$R \perp S \equiv R \mid S \mid R^{-1}$$

This has the following property: $R \perp S$ relates x to y if and only if there are u and v such that R relates x to u and y to v, and S relates u to v. That is, $<x, y> \in R \perp S$ if and only if there are u and v such that $<x, u> \in R$, $<y, v> \in R$ and $<u, v> \in S$. This can be visualized:


$$\begin{tikzcd}
x \arrow[r, "R"] \arrow[d, dotted, "R \perp S"'] & u \arrow[d, "S"] \\
y \arrow[r, "R"'] & v
\end{tikzcd}$$

Now, if we have that R_i relates inequivalent states, then $(T \downarrow a) \perp R_i$ will relate those states that are carried by symbol a into states inequivalent by R_i. Thus, we define R_{i+1} so that it relates those states that are related by R_i together with those related by $(T \downarrow a) \perp R_i$, for any $a \in \Sigma$. Now, if by $[\perp R_i]$ we mean the function that takes a polymorphic image of R_i, that is,

$$[\perp R_i]\, x = x \perp R_i$$

then it is easy to see that

$$(T \mid [\perp R_i]) \downarrow a = [\perp R_i]\,(T \downarrow a) = (T \downarrow a) \perp R_i$$

Hence, the union of $(T \downarrow a) \perp R_i$, for all $a \in \Sigma$, is just the union of the range of the relation $T \mid [\perp R_i]$. This yields the definition of R_i and hence ψ:

$$R_{i+1} \equiv \psi R_i \equiv R_i \cup \bigcup(\textbf{rng } (T \mid [\perp R_i]))$$

This completes the definition of the inequivalence relation R_∞. Two states are now equivalent if they are not inequivalent:

$$R_= \equiv Q^2 \setminus (R_\infty \cup R_\infty^{-1})$$

where Q^2 means $Q \times Q$.

The minimal machine is constructed on the basis of the equivalence classes of states under $R_=$. The equivalence class of a state q is just the unit image under $R_=$ of q:

$$\text{eclass } q \equiv R_{=} \textbf{ unimg } q$$

The set of all such equivalence classes results from taking the image of Q under eclass:

$$Q_{=} \equiv \text{eclass } \textbf{img } Q$$

In general, we define

$$\text{equiv} \equiv [\text{eclass } \textbf{img}],$$

so $Q_{=} \equiv \text{equiv } Q$. We take $Q_{=}$ to be the states in the minimal machine.

It remains to construct the transition relation $T_{=}$ of the minimal machine. For all $<q, q'> \in T \downarrow a$ we want

$$<\text{eclass } q, \text{eclass } q'> \in T_{=} \downarrow a$$

Thus $T_{=} \downarrow a$ is the *isomorphic image* under eclass of $T \downarrow a$:

$$T_{=} \downarrow a = \text{eclass } \$ \ (T \downarrow a)$$

Thus $T_{=}$ is the (finite) composition of $[\text{eclass } \$]$ and T, which is the (finite) relative product of T and $[\text{eclass } \$]$:

$$T_{=} \equiv T \mid [\text{eclass } \$]$$

The remainder of the minimal machine is easy to construct. For example, the final states are just the equivalence classes of the original final states:

$$F_{=} \equiv \text{equiv } F$$

There follows the actual relational program to minimize a small automaton. It makes use of two auxiliary functions σ and ρ for defining the union of a set of sets:

-- DFA Minimization

-- Utility Functions

$$\begin{aligned}
\text{1st} &\equiv [\downarrow 1] \\
\text{2nd} &\equiv [\downarrow 2] \\
r \perp s &\equiv r \mid s \mid r^{-1} \\
\sigma\, f &\equiv (f \circ (\text{1st} \,\overline{,}\, (\epsilon \circ \text{2nd})))\,\overline{,}\,([\backslash] \circ (\mathbf{I} \,\overline{,}\, (\mathbf{un} \circ \epsilon)) \circ \text{2nd}) \\
f \;\rho\; i &\equiv \text{1st} \circ (\sigma\, f \textbf{ while } ([\neq \emptyset] \circ \text{2nd})) \circ [i\,,] \\
\bigcup &\equiv [\cup]\; \rho\; \emptyset
\end{aligned}$$

-- Example DFA

$$\begin{aligned}
\Sigma &\equiv \{1, 2\} \\
T &\equiv \{1 : \{10 : 10, 20 : 20\}, 2 : \{10 : 30, 20 : 30\}\} \\
Q &\equiv \{10, 20, 30\} \\
F &\equiv \{30\}
\end{aligned}$$

-- Minimization

$$\begin{aligned}
Q^2 &\equiv Q \times Q \\
n &\equiv \textbf{size } Q^2 \\
R_0 &\equiv F \times (Q \setminus F) \\
\psi\ R &\equiv R \cup \bigcup(\textbf{rng } (T \mid [\perp R])) \\
R_\infty &\equiv \psi^n\ R_0 \\
R_= &\equiv Q^2 \setminus (R_\infty \cup R_\infty^{-1}) \\
\text{eclass} &\equiv [R_= \textbf{ unimg}] \\
\text{equiv} &\equiv [\text{eclass } \textbf{img}] \\
Q_= &\equiv \text{equiv } Q \\
T_= &\equiv T \mid [\text{eclass } \$] \\
F_= &\equiv \text{equiv } F
\end{aligned}$$

-- Minimized DFA

val $Q_=$
{ {10, 20}, {30} }
val $T_=$
{ 1 : { {10, 20} : {10, 20} },
2 : { {10, 20} : {30} } }
val $F_=$
{ {30} }

The **val** command prints the value of an identifier.

3. Gaussian Elimination

The matrix is represented as a vector of vectors:

$$M = \begin{array}{l} << a_{11}\ , \ldots ,\ a_{1n},\ b_1\ >, \\ \ < a_{21}\ , \ldots ,\ a_{2n},\ b_2\ >, \\ \quad \vdots \qquad\qquad \vdots \quad \vdots \\ \ < a_{n1}\ , \ldots ,\ a_{nn},\ b_n\ >> \end{array}$$

For the sake of simplicity we assume all the a_{ij} are nonzero. We use the operator '$\downarrow$' to select the kth element of a vector. Thus '$M \downarrow k$' is the kth row of M and '$M \mid [\downarrow k]$' is the kth column of M.

The Gauss Elimination function will use n successive steps. Each of these steps will accomplish the transformation

$$<M, k> \rightarrow <M', k+1>$$

where M' is obtained from M by performing the elimination process on the kth column:

$$M' = \text{elim } <M, k>$$

Thus the complete process is defined:

$$\text{Gauss } M = (\text{elim for } <1\ , \ldots ,\ n>)\ M$$

Here we make use of the functional '$(f$ for $S)\ x$' which computes the sequence of values

$$\begin{aligned}
y_1 &= f<x, S_1> \\
y_2 &= f<y_1, S_2> \\
&\ \vdots \\
y_n &= f<y_{n-1}, S_n>
\end{aligned}$$

and returns y_n. The 'for' functional is defined in terms of reduction as follows:

$$f \text{ for } S \equiv [@ \; S] \circ [f \; \S]$$

This can be understood by the expansion:

$$\begin{aligned}(f \text{ for } S)\; x &= ([@ \; S] \circ [f \; \S])\; x = [@ \; S]\; ([f \; \S]\; x) \\ &= [@ \; S]\; (f \; \S \; x) = (f \; \S \; x) \; @ \; S \\ &= (f \; \S \; x)\; S\end{aligned}$$

We turn now to the elimination process. We want $M' = \text{elim} <M, k>$, where M' results from M by zeroing all entries in column k of M, except the entry in row k, which is set to one. This can be accomplished by subtracting an appropriate matrix E from M:

$$M' = \text{matdif} <M, E>$$

Here 'matdif' is a component-wise matrix difference function.

The matrix E is produced by multiplying the appropriate factors by the individual rows of M. For the first elimination step E is:

$$E_1 = \begin{array}{l} < (1-1/a_{11})M_1, \\ \quad (a_{21}/a_{11})M_1, \\ \quad \vdots \\ \quad (a_{n1}/a_{11})M_1 \; > \end{array}$$

The matrix resulting from subtracting E_1 from M is

$$M_1 = \begin{array}{llllll} << 1, & a_{12}', & \cdots & a_{1n}', & b'_1 & >, \\ < 0, & a_{22}', & \cdots & a_{2n}', & b'_2 & >, \\ \;\;\vdots & \vdots & & \vdots & \vdots & \\ < 0, & a_{n2}', & \cdots & a_{nn}', & b'_n & >> \end{array}$$

At the next stage the elimination matrix is:

$$E_2 = \begin{array}{l} < (a_{12}'/a_{22}')M'_2, \\ \quad (1-1/a_{22}')M_2', \\ \quad \vdots \\ \quad (a_{n2}'/a_{22}')M_2' \; > \end{array}$$

In general, if M'' is the matrix resulting from the $k-1$st elimination step, then the elimination matrix for the kth step is

$$E_k = \begin{array}{l} < (a_{1k}''/a_{kk}'')M_k'', \\ \quad \vdots \\ \quad ([a_{kk}-1]/a_{kk}'')M_k'', \\ \quad \vdots \\ \quad (a_{nk}''/a_{kk}'')M_k'' \; > \end{array}$$

It is easy to see that E_k results from multiplying a vector V_k by the kth row of M''. This is just the outer product of V_k and the kth row of M'':

$$E_k = \text{outerprod} < V_k, M'' \downarrow k>$$

The vector V_k is

$$V_k = \begin{array}{l} < \quad a_{1k}''/a_{kk}'', \\ \qquad \vdots \\ \quad (a_{kk}''-1)/a_{kk}'', \\ \qquad \vdots \\ \quad a_{nk}''/a_{kk}'' \quad > \end{array}$$

This is obtained by forming the scalar product of $1/a_{kk}''$ and the vector

$$U_k = <a_{1k}'', a_{2k}'', \ldots, a_{kk}''-1, \ldots, a_{nk}''>$$

This in turn is the result of subtracting from the kth column of M'' the *unit vector* unit $<M, k>$, which has a 1 in the kth position, and a 0 in all others.

We now develop an explicit relational formula for V_k. For this purpose it will be convenient to treat it as a binary function $V_k = V<M, k>$. We have:

$$\begin{aligned} V<M, k> &= \text{scaprod} <1 \,/\, \text{diag} <M, k>, U_k> \\ &= \text{scaprod} <1 \,/\, \text{diag} <M, k>, \text{vecdif} <\text{column} <M, k>, \text{unit} <M, k>>> \end{aligned}$$

The parameter $<M, k>$ can be factored out by use of the construction operation '$\overline{,}$', which is defined so that $(f \overline{,} g)x = <fx, gx>$. Factoring, we have:

$$\begin{aligned} V<M, k> &= \text{scaprod} <1 \,/\, \text{diag} <M, k>, \text{vecdif} <\text{column} <M, k>, \text{unit} <M, k>>> \\ &= \text{scaprod} <([1\,/] \circ \text{diag}) <M, k>, \text{vecdif} ((\text{column} \overline{,} \text{unit}) <M, k>)> \\ &= \text{scaprod} <([1\,/] \circ \text{diag}) <M, k>, (\text{vecdif} \circ (\text{column} \overline{,} \text{unit})) <M, k>> \\ &= \text{scaprod} ((([1\,/] \circ \text{diag}) \overline{,} (\text{vecdif} \circ (\text{column} \overline{,} \text{unit}))) <M, k>) \\ &= (\text{scaprod} \circ (([1\,/] \circ \text{diag}) \overline{,} (\text{vecdif} \circ (\text{column} \overline{,} \text{unit}))) <M, k> \end{aligned}$$

Canceling $<M, k>$ from both sides yields an explicit formula for V:

$$V \equiv \text{scaprod} \circ (([1\,/] \circ \text{diag}) \overline{,} (\text{vecdif} \circ (\text{column} \overline{,} \text{unit})))$$

We proceed similarly to get a formula for elim:

$$\begin{aligned} \text{elim} <M, k> &= \text{matdif} <M, E_k> \\ &= \text{matdif} <M, \text{outerprod} <V_k, M \downarrow k>> \\ &= \text{matdif} <M, \text{outerprod} <V <M, k>, M \downarrow k>> \end{aligned}$$

We perform some minor rearrangements so that $<M, k>$ can be factored out of the right-hand side:

$$\begin{aligned} \text{elim} <M, k> &= \text{matdif} <M, \text{outerprod} <V <M, k>, [\downarrow] <M, k>>> \\ &= \text{matdif} <M, (\text{outerprod} \circ (V \overline{,} [\downarrow])) <M, k>> \\ &= \text{matdif} <[\downarrow 1] <M, k>, (\text{outerprod} \circ (V \overline{,} [\downarrow])) <M, k>> \\ &= (\text{matdif} \circ ([\downarrow 1] \overline{,} (\text{outerprod} \circ (V \overline{,} [\downarrow])))) <M, k> \end{aligned}$$

Canceling $<M, k>$ from both sides yields an explicit formula for elim:

$$\text{elim} \equiv \text{matdif} \circ ([\downarrow 1] \overline{,} (\text{outerprod} \circ (V \overline{,} [\downarrow])))$$

A complete RPL session demonstrating the Gaussian elimination function follows:

```
-- Utility Functions

         con k  ≡  λ x k
    transmap f  ≡  [| f] ∘ [#]
        vecdif  ≡  transmap [−]
 scaprod <k, v> ≡  v | [k ×]
outerprod <u, v> ≡  u | (scaprod ∘ [, v])
        matdif  ≡  transmap vecdif
  column <M, k> ≡  M | [↓ k]
    unit <M, k> ≡  <1 , . . . , size M> | [[= k] → con 1: con 0]
    diag <M, k> ≡  M ↓ k ↓ k
       f for S  ≡  [@ S] ∘ [f §]

-- Gaussian Elimination

             V  ≡  scaprod ∘ (([1.0 /] ∘ diag)‾,(vecdif ∘ (column‾,unit)))
          elim  ≡  matdif ∘ ([↓ 1]‾,(outerprod ∘ (V‾,[↓] )))
       Gauss M  ≡  (elim for <1 , . . . , size M>) M

-- Example Matrix

             M  ≡  <<3, 9, 33>, <2, −1, 1>>

-- Execution

   Gauss M

     << 1.0,  −2.38419E−7,  2.0>
      < 0.0,          1.0,  3.0>>
```

The matrix M represents the equations

$$\begin{aligned} 3x + 9y &= 33 \\ 2x - y &= 1 \end{aligned}$$

The result of Gauss M correctly reflects the solution $x = 2$, $y = 3$.

4. Employee File Update

Next we consider a simple data processing example adapted from [MacLennan83]. We are given an employee file F indexed by employee number. That is, $F \downarrow n$ is the record for employee number n. The employee records themselves are represented by functions from attribute names into attribute values. For example, if R is an employee record, then $R \downarrow$ "N" is the employee's name, $R \downarrow$ "R" is his hourly rate, and $R \downarrow$ "H" is the hours worked this pay period. Here is an example employee file containing three records:

```
F  ≡  {124 : {"N" : "John", "R" : 10, "H" : 100},
       118 : {"N" : "Bill", "R" : 15, "H" : 120},
       207 : {"N" : "Sally", "R" : 14, "H" : 115}}
```

We are also given an update file U such that $U \downarrow n$ is the number of hours worked this week by employee number n. For example:

```
U  ≡  {118 : 6, 124 : 40, 207 : 40}
```

Our task is to generate an updated employee file F' in which the hours worked ("H") field has been updated.

First we define 'sumhrs' so that if R is an employee record and h is the hours worked this week, then sumhrs $<R, h>$ is the new total hours. Clearly,

$$\text{sumhrs} <R, h> \equiv (R \downarrow \text{“H”}) + h$$

Alternately, we can define this function variable-free style:

$$\text{sumhrs} \equiv [+] \circ ([\downarrow \text{“H”}] \,\|\, \mathbf{I})$$

It is easy to see the two are equivalent:

$$\begin{aligned}
\text{sumhrs} <R, h> &= ([+] \circ ([\downarrow \text{“H”}] \,\|\, \mathbf{I})) <R, h> \\
&= [+] (([\downarrow \text{“H”}] \,\|\, \mathbf{I}) <R, h>) \\
&= [+] <[\downarrow \text{“H”}] \, R, \mathbf{I} \, h> \\
&= (R \downarrow \text{“H”}) + h
\end{aligned}$$

Our next task is to replace the old value of “H” field by h'. This can be accomplished by the *ordered union* operation ‘;’. For example,

$$\{\text{“H”} : h'\} \, ; R$$

will return a record R' in which $R' \downarrow \text{“H”} = h'$ but all other fields of R' are the same as in R. How do we get the relation $\{\text{“H”} : h'\}$? Since this is just a sequence that's equivalent to the array $<\text{“H”}, h'>$, we can use **as** to convert the array to a sequence. We solve for the function f that computes $\{\text{“H”} : h'\}$ from $<R, h>$ as follows:

$$\begin{aligned}
f <R, h> &= \{\text{“H”} : \text{sumhrs} <R, h>\} \\
&= \mathbf{as} <\text{“H”}, \text{sumhrs} <R, h>> \\
&= \mathbf{as} \, ([\text{“H”} ,] \, (\text{sumhrs} <R, h>)) \\
&= (\mathbf{as} \circ [\text{“H”} ,] \circ \text{sumhrs}) <R, h>
\end{aligned}$$

Hence,

$$f \equiv \mathbf{as} \circ [\text{“H”} ,] \circ \text{sumhrs}$$

It's necessary to get the corresponding records from the F and U files together so that they can be processed by f. This is accomplished by the extensional construction operation $\#$ defined so that $(F \# U) \downarrow n = <F \downarrow n, U \downarrow n>$. With the given example files we have:

$$\begin{aligned}
F \# U = \{ \, & 124 : <\{\text{“N”} : \text{“John”}, \text{“R”} : 10, \text{“H”}: 100\}, 40>, \\
& 118 : <\{\text{“N”} : \text{“Bill”}, \text{“R”} : 15, \text{“H”} : 120\}, 6>, \\
& 207 : <\{\text{“N”} : \text{“Sally”}, \text{“R”} : 14, \text{“H”} : 115\}, 40>\}
\end{aligned}$$

Notice that the pairs $<F \downarrow n, U \downarrow n>$ are just the inputs required for f. We combine the preceding results into a update file ‘upd’ defined so that upd $\downarrow n$ is $\{\text{“H”} : h'\}$, representing the new hours worked for employee number n. In this case,

$$\begin{aligned}
\text{upd} = \{ \, & 124 : \{\text{“H”} : 140\}, \\
& 118 : \{\text{“H”} : 126\}, \\
& 207 : \{\text{“H”} : 155\}\}
\end{aligned}$$

It's easy to solve for upd by using the relative product:

$$\begin{aligned}
\text{upd} \downarrow n &= f \, ((F \# U) \downarrow n) \\
&= ((F \# U) \,|\, f) \downarrow n
\end{aligned}$$

Hence, upd $= (F \# U) \,|\, f$. Substituting for f yields:

$$\text{upd} \equiv (F \# U) \,|\, (\mathbf{as} \circ [\text{“H”} ,] \circ \text{sumhrs})$$

Now we're almost done. We want each record in F' to be the ordered union of the corresponding update record in upd and old record in F. Hence we solve:

$$\begin{aligned} F' \downarrow n &= (\text{upd} \downarrow n) \,;\, (F \downarrow n) \\ &= [;] <\text{upd} \downarrow n, F \downarrow n> \\ &= [;] ((\text{upd} \# F) \downarrow n) \\ &= ((\text{upd} \# F) \mid [;]) \downarrow n \end{aligned}$$

Hence,

$$F' = (\text{upd} \# F) \mid [;]$$

The complete session follows:

```
-- The Files
    F ≡ {124 : {"N" : "John", "R" : 10, "H" : 100},
         118 : {"N" : "Bill", "R" : 15, "H" : 120},
         207 : {"N" : "Sally", "R" : 14, "H" : 115}}

    U ≡ {118 : 6, 124 : 40, 207 : 40}
-- Computing the New File
   sumhrs ≡ [+] ∘ ([↓ "H"] || I)

      upd ≡ (F # U) | (as ∘ ["H" ,] ∘ sumhrs)

       F′ ≡ (upd # F) | [;]
-- The New File
   val F′

    {124 : {"H" : 140, "N" : "John", "R" : 10},
     118 : {"H" : 126, "N" : "Bill", "R" : 15},
     207 : {"H" : 155, "N" : "Sally", "R" : 14} }
```

This result correctly reflects the fact that John (employee 124) has worked 124 hours, Bill (employee 118) has worked 15 hours, and Sally (employee 207) has worked 14 hours.

It is simple to modify the program so that it uses the input files OldMaster and Updates, and defines the output file NewMaster:

```
        F ≡ file "OldMaster"
        U ≡ file "Updates"
   sumhrs ≡ [+] ∘ ([↓ "H"] || I)
      upd ≡ (F # U) | (as ∘ ["H" ,] ∘ sumhrs
     file "NewMaster" ≡ (upd # F) | [;]
```

5. References

[Brown&Mitton] Brown, J. R., and Mitton, S. J., *Relational Programming: Design and Implementation of a Prototype Interpreter*, MS thesis, Naval Postgraduate School, June 1985.

[MacLennan83] MacLennan, B. J., "Relational Programming," Naval Postgraduate School Computer Science Department Technical Report NPS52-83-012, September 1983.

APPENDIX A: EXAMPLE RPL SESSIONS

This appendix contains transcripts of actual RPL sessions with the Brown and Mitton interpreter. Note that the interpreter follows the Interlisp convention of permitting a bracket ']' to close any number of open parentheses.

Example 1: Word Frequence

It will be seen that the RPL interpreter computes a relation containing redundant tuples. They do no harm, but can be eliminated (by a quadratic algorithm) if desired. The transcript follows:

```
Loading RPL--- DO YOU WANT TO RESUME A PREVIOUS RPL SESSION? <y/n> n


RPL INTERPRETER ON LINE!!

?> S == (list "to" "be" "or" "not" "to" "be"]

?> (S sup -1]

 ( rel  ( be  6 )  ( to  5 )  ( not  4 )
  ( or  3 )  ( be  2 )  ( to  1 ) )

?> unimage T == ((dom T) restrict (lsec T unimg]

?> (unimage (S sup -1]

 ( set  ( be  ( set  6  2 ) )  ( to  ( set  5  1 ) )
  ( not  ( set  4 ) )  ( or  ( set  3 ) )
  ( be  ( set  6  2 ) )  ( to  ( set  5  1 ) ) )

?> ((unimage (S sup -1)) rp size]

 ( rel  ( be  2 )  ( to  2 )  ( not  1 )  ( or  1 )
  ( be  2 )  ( to  2 ) )

?> freq S == ((unimage (S sup -1)) rp size]

?> (freq (list "to" "be" "or" "not" "to" "be"]

 ( rel  ( be  2 )  ( to  2 )  ( not  1 )  ( or  1 )
  ( be  2 )  ( to  2 ) )

?> done

DO YOU WANT TO SAVE ENVIRONMENT FOR FUTURE USE? <y/n> n
```

Example 2: Minimizing DFA

For this example we assume that commands for defining the DFA and performing the minimization are on a file, "examples/dfa.rpl", whose contents are:

```
(1st == (rsec sel 1))
(2nd == (rsec sel 2))
(r ppd s == (r | (s | (cnv r))))
(sigma f == ((f o (1st (, bar) (epsilon o 2nd))) (, bar) ((op ) o ((I (, bar) (un o epsilon)) o 2nd))))
(f rho i == (1st o (((sigma f) while ((rsec != empty) o 2nd)) o (lsec i ,))))
(union == ((op cup) rho empty))
(SIGMA == (set 1 2))
(T == (rel (1 : (rel (10 : 10) (20 : 20))) (2 : (rel (10 : 30) (20 : 30)))))
(Q == (set 10 20 30))
(F == (set 30))
(Q_sup_2 == (Q cart Q))
(n == (size Q_sup_2))
(R_sub_0 == (F cart (Q  F)))
(psi R == (R cup (union (rng (T rp (rsec ppd R))))))
(R_sub_inf == ((psi sup n) R_sub_0))
(R_sub_= == ((Q_sup_2  R_sub_inf) cap (Q_sup_2  (cnv R_sub_inf))))
(rom_eclass == (lsec R_sub_= unimg))
(rom_equiv == (lsec rom_eclass img))
(Q_sub_= == (rom_equiv Q))
(T_sub_= == (T rp (lsec rom_eclass $)))
(F_sub_= == (rom_equiv F))
EOF
```

This file is executed by being loaded into RPL. The resulting transition function and states of the minimal machine are then displayed. They can be seen to be sets of sets, since the states in the minimal machine are represented by equivalence classes.[3] The transcript follows:

```
DO YOU WANT TO RESUME A PREVIOUS RPL SESSION? <y/n> y
INPUT FILENAME
examples/dfa.rpl
Loading--- Session loaded
?> val Q_sub_=

 ( set  ( set  10  20 )  ( set  10  20 )  ( set  30 ) )

?> val T_sub_=

 ( rel  ( 1  ( rel  ( ( set  10  20 )  ( set  10  20 ) )  ( ( set  10  20 )  (
 set  10  20 ) ) ) )  ( 2  ( rel  ( ( set  10  20 )  ( set  30 ) )  ( ( set  10
 20 )  ( set  30 ) ) ) ) )

?> val F_sub_=

 ( set  ( set  30 ) )

?> done
```

3. Note that as usual there is benign redundancy in the sets.

Example 3: Gaussian Elimination

The program for performing the Gaussian elimination is in the file "examples/gauss.rpl", whose contents are:

```
(con k == (func x k))
(transmap f == ((rsec rp f) o (op #)))
(vecdif == (transmap (op -)))
(scaprod (k v) == (v rp (lsec k times)))
(outerprod (u v) == (u rp (scaprod o (rsec . v))))
(matdif == (transmap vecdif))
(column (M k) == (M rp (rsec sel k)))
(unit (M k) == ((listrange 1 to (size M)) rp (if (rsec = k) -> (con 1) : (con 0))))
(diag (M k) == ((M sel k) sel k))
(f for S == ((rsec @ S) o (lsec f red)))
(V == (scaprod o (((lsec 1.0 divide) o diag) (. bar) (vecdif o (column (. bar) unit)))))
(elim == (matdif o ((rsec sel 1) (. bar) (outerprod o (V (. bar) (op sel) )))))
(Gauss M == ((elim for (listrange 1 to (size M))) M))
(M == (list (list 3 9 33) (list 2 -1 1)))
(a == (diag (list M 1)))
(b == (vecdif (list (column (list M 1)) (unit (list M 1)))))
(v == (scaprod (list 0.33 b)))
EOF
```

The session shown in the following transcript performs the Gaussian elimination on the matrix M:

```
DO YOU WANT TO RESUME A PREVIOUS RPL SESSION? <y/n> y
INPUT FILENAME
examples/gauss.rpl
Loading--- Session loaded
?> (Gauss M]

 ( rel ( 1 ( rel ( 1 1.0 ) ( 2 -2.38419E-07 ) ( 3 2.0 ) ) )
  ( 2 ( rel ( 1 0.0 ) ( 2 1.0 ) ( 3 3.0 ) ) ) )

?> done
```

Note that the resulting matrix is printed as a relation rather than a list of lists, since it is quite expensive for the interpreter to determine if a relation is in fact a list.

Example 4: Data Processing

In this example, the employee file to be updated is small (three records), and so typed in interactively. More typically, the RPL **file** facility would be used to load *F* from disk. The transcript follows:

```
RPL INTERPRETER ON LINE!!
?> F == (rel (124 :
 (rel ("N" : "John") ("R" : 10) ("H" : 100)))
(118 :
 (rel ("N" : "Bill") ("R" : 15) ("H" : 120)))
(207 :
 (rel ("N" : "Sally") ("R" : 14) ("H" : 115]

?> U == (rel (118 : 6) (124 : 40) (207 : 40]

?> (F # U]

 ( rel  ( 124  ( rel
   ( 1  ( rel  ( N  John )  ( R  10 )  ( H  100 ) ) )
   ( 2  40 ) ) )
  ( 118  ( rel
   ( 1  ( rel  ( N  Bill )  ( R  15 )  ( H  120 ) ) )
   ( 2 6 ) ) )
  ( 207  ( rel
   ( 1  ( rel  ( N  Sally )  ( R  14 )  ( H  115 ) ) )
   ( 2 40 ) ) ) )
?> sumhrs == ((op +) o ((rsec sel "H") || I]

?> upd == ((F # U) rp
    (as o ((lsec "H" .) o sumhrs]

?> F' == ((upd # F) rp (op :]

?> val F'

 ( rel  ( 124
   ( rel  ( H  140 )  ( N  John )  ( R  10 ) ) )
  ( 118
   ( rel  ( H 126 )  ( N  Bill )  ( R  15 ) ) )
  ( 207
   ( rel  ( H  155 )  ( N  Sally )  ( R 14 ) ) ) )

?> done
```

APPENDIX B: RPL GRAMMAR

$$session \;=\; command^{*}\ \mathbf{done}$$

$$command \;=\; \left\{\begin{array}{l} prefixid\ [identifier] \equiv expression \\ \mathbf{display}\ expression \end{array}\right\}$$

$$expression \;=\; \left\{\begin{array}{l} [expression\ infix]\ application \\ superscription \end{array}\right\}$$

$$application \;=\; \left\{\begin{array}{l} [application]\ primary \\ \mathbf{iter}\ \text{'['}\ primary \rightarrow primary\ \text{']'} \end{array}\right\}$$

$$superscription \;=\; expression\ \mathbf{sup} \left\{\begin{array}{l} application \\ + \\ * \end{array}\right\}$$

$$primary \;=\; \left\{\begin{array}{l} literal \\ prefixid \\ \text{'['} \left\{\begin{array}{l} infix \\ infix\ primary \\ primary\ infix \\ primary \rightarrow primary\ ;\ primary \end{array}\right\} \text{']'} \\ (\ expression\ [..\ expression]\ ) \\ \{\ expression\ [..\ expression]\ \} \\ <\ primary\ ,\ \cdots\ > \\ \mathbf{file}\ string \end{array}\right\}$$

$$infix \;=\; infixop\ [\mathbf{bar}]$$

$$identifier \;=\; letter \left[\begin{array}{l} letter \\ digit \end{array}\right]^{*} prime^{*}$$

$$prime \;=\; '$$

$$literal \;=\; \left\{\begin{array}{l} digit^{+}\ [.\ digit^{+}]\ [\mathbf{E}\ [\ +\ |\ -\ ]\ digit^{+}] \\ string \\ \mathbf{true} \\ \mathbf{false} \end{array}\right\}$$

$$string \;=\; \text{``}\ char^{*}\ \text{''}$$

infixop =

```
sel | . : cup member nomem !subset subset = -> <- restr ; cl cr cap \
@hat ! cat @ . || $ red + - times divide != < > <= >=
andsign orsign cart
```

$$\textit{prefixid} = \begin{Bmatrix} \textit{identifier} \\ \textit{prefixop} \end{Bmatrix}$$

prefixop =

```
- un cur unc theta size str DELTA inv dom rng mem Lm Rm Mm run lun bun
init term alpha omega ALPHA OMEGA min max mu index select join as sa sa0
rp rpi rsort sort unimg all ssm img curry uncurry PHI Id while upsilon
phi delta PI extend restrict wig not
```

APPENDIX C: RPL INPUT FORM SUMMARY

TABLE 1. Primitive Extensional Operations

Name	Old Input Form	New Input Form	Publication Form
selection	t sel x	t sel x	$t \downarrow x$
relative product	t | u	t | u	$t \mid u$
construction	t , bar u	t # u	$t \# u$
pair formation	x : y	x : y	$x : y$
union	t cup u	t cup u	$t \cup u$
unit set	un x	un x	**un** x
currying	cur t	cur t	**cur** t
uncurrying	unc t	unc t	**unc** t
unique element selection	theta s	theta s	$\theta\ s$
element selection	(added)	epsilon t	$\epsilon\ t$
cardinality	size t	size t	size t
structure	str t	(deleted)	(deleted)
transitive closure	t sup +	t sup +	t^+
empty set	empty	empty	$\emptyset$

TABLE 2. Nonprimitive Extensional Operations: Group 1

Name	Old Input Form	New Input Form	Publication Form
pair list	(x, y)	(x , y)	(x, y)
left pair section	(x,)	(deleted)	(deleted)
right pair section	(,y)	(deleted)	(deleted)
duplication	DELTA x	DELTA x	$\Delta\ x$
membership	x member t	x member t	$x \in t$
nonmembership	x nomem t	x nomem t	$x \notin t$
improper subset	s !subset t	s !subset t	$s \subseteq t$
proper subset	s subset t	s subset t	$s \subset t$
equality	s = t	s = t	$s = t$
converse	inv t, t sup -1	cnv t, t sup -1	**cnv** t, t^{-1}
domain	dom t	dom t	**dom** t
range	rng t	rng t	**rng** t
members	mem t	mem t	**mem** t
left member	Lm (x,t)	x Lm t	x **Lm** t
right member	Rm (x,t)	x Rm t	x **Rm** t
member	Mm (x,t)	x Mm t	x **Mm** t
right univalent	run t	run t	**run** t
left univalent	lun t	lun t	**lun** t
bi-univalent	bun t	bun t	**bun** t
initial members	init t	init t	**init** t
terminal members	term t	term t	**term** t
reflexive transitive closure	t sup *	t sup **	t^*
domain restriction	p -> t	p -> t	$p \rightarrow t$
range restriction	t <- p	t <- p	$t \leftarrow p$
restriction	t restr p	t restr p	$t \upharpoonright p$
sequence filtering	(added)	p xi t	$p\ \xi\ t$

TABLE 3. Nonprimitive Extensional Operations: Group 2

Name	Old Input Form	New Input Form	Publication Form
first member	alpha t	alpha t	$\alpha\ t$
last member	omega t	omega t	$\omega\ t$
initial sequence	ALPHA t	ALPHA t	$\mathrm{A}\ t$
final sequence	OMEGA t	OMEGA t	$\Omega\ t$
ordered union	t ; u	t ; u	$t\ ;\ u$
cons left	x cl t	x cl t	x **cl** t
cons right	t cr x	t cr x	t **cr** x
minimum	min s	min s	**min** s
maximum	max s	max s	**max** s
intersection	s cap t	s cap t	$s \cap t$
set difference	s \ t	s \ t	$s \setminus t$
apply functional record	t @ hat x	t @hat x	$t\ @\ x$
apply functional structure	t ! x	t ! x	$t\ !\ x$
minimize	mu t	mu t	$\mu\ t$
database index	index x d	x index d	x **index** d
database select	select x	x select d	x **select** d
database join	join x	x join dblist	x **join** $dblist$
array to sequence	as t	as t	**as** t
sequence to array	sa t	t sa i	t **sa** i
seq. to zero-origin array	sa0 t	(deleted)	(deleted)
relative product	rp f t	t rp f	$t \mid f$
relative product inverse	rpi f t	f rpi t	$f \mid t$
array concatenation	t cat u	t cat u	t **cat** u
relation sort	rsort s	rsort s	**rsort** s
sort	sort s	sort s	**sort** s
unit image	unimg t x	t unimg x	t **unimg** x
all	all t	all t	**all** t
sequence to matrix	ssm t	ssm t	**ssm** t

TABLE 4. Primitive Intensional Operations

Name	Old Input Form	New Input Form	Publication Form
application	f @ x	f @ x	$f\ @\ x$
image	img f s	f img s	f **img** s
composition	f . g	f o g	$f \circ g$
infix to prefix	(added)	(op +), (op times), ...	$[+], [\times], \cdots$
left section	(x+), (x-), ...	(lsec x +), (lsec x -), ...	$[x+], [x-], \cdots$
right section	(+y), (-y), ...	(rsec + y), (rsec - y), ...	$[+y], [-y], \cdots$
paralleling	f || g	f || g	$f \parallel g$
isomorphism	f $ t	f $ t	$f\ \$\ t$
formal application	f @ bar g	(deleted)	(deleted)
functional condition	(p -> f; g)	(if p -> f ; g)	$(p \rightarrow f; g)$
curry	curry f	curry f	**curry** f
uncurry	uncurry f	uncurry f	**uncurry** f
filtering	PHI p (d, r)	p PHI S	$p\ \Phi\ S$
iteration	iter [p -> f]	(iter p -> f)	**iter** $[p \rightarrow f]$
formalization	+ bar, times bar, ...	(+ bar), (times bar), ...	$\overline{+}, \overline{\times}, \ldots$
identity	Id	I	**I**

TABLE 5. Nonprimitive Intensional Operations

Name	Old Input Form	New Input Form	Publication Form
while loop	while [p, f]	(f while p)	f **while** p
array reduction	f red i	f red x	$f \S x$
repeated composition	f sup n	f sup n	f^n
value of node	upsilon f	upsilon f	υf
operate on form	phi f	phi f	ϕf
operate on data	delta f	delta f	δf
image of structure	PI f	PI f	Πf
extension	extend (t, f)	t extend f	t **extend** f
restriction	restrict (s, f)	s restrict f	s **restrict** f
formal negation	wig p	wig p	$\sim p$

TABLE 6. Miscellaneous Operations

Name	Old Input Form	New Input Form	Publication Form
sum	x + y	x + y	$x + y$
difference	x - y	x - y	$x - y$
product	x times y	x times y	$x \times y$
quotient	x divide y	x divide y	x / y
inequality	x != y	x != y	$x \neq y$
less	x < y	x < y	$x < y$
greater	x > y	x > y	$x > y$
less or equal	x <= y	x <= y	$x \leqslant y$
greater or equal	x >= y	x >= y	$x \geqslant y$
conjunction	x andsign y	x andsign y	$x \wedge y$
disjunction	x orsign y	x orsign y	$x \vee y$
negation	not x	not x	$\neg x$
cartesian product	s cart t	s cart t	$s \times t$

TABLE 7. Data Input Operations and Syntax

Name	Input Form	Publication Form
identifiers	a, b', total, etc.	a, b', total, *etc.*
strings	"abcd"	"abcd"
booleans	true, false	**true**, **false**
relation	(rel (x : y), ...)	$((x\ y),\ \cdots)$
set	(set x y ...)	$\{x,\ y,\ \cdots\}$
sequence	(seq x y ...)	$(x,\ y,\ \cdots)$
list	(list x y ...)	$<x,\ y,\ \cdots>$
subrange set	(setrange m to n)	$\{m, \ldots, n\}$
subrange sequence	(seqrange m to n)	$(m, \ldots, n)$
subrange list	(listrange m to n)	$<m, \ldots, n>$

TABLE 8. RPL Command Types

Name	Input Form	Publication Form
data definition	x == y	$x \equiv y$
prefix function definition	f x == y	$f\ x \equiv y$
infix function definition	x f y == z	$x\ f\ y \equiv z$
write data to a file	file "name" == x	**file** *"name"* $\equiv x$
read data from a file	x == (file "name")	$x \equiv$ **file** *"name"*
output, form 1	display x	**display** x
output, form 2	dis x	**display** x
output, form 3	d x	**d** x
output, form 4	x	x
output value of definition	val x	**val** x
output function environment	env f	**env** f
output entire environment	env	**env**

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943	40
Bruce J. MacLennan Code 52ML Department of Computer Science Naval Postgraduate School Monterey, CA 93943	12
Ralph Wachter Code 433 Office of Naval Research 800 N. Quincy Arlington, VA 22217-5000	1
S. Kamal Abdali Tektronix Laboratories Computer Research Laboratory M/S 50-662 P. O. Box 500 Beaverton, OR 97077	1
Drew D. Adams Centre de Recherches de la C.G.E. Laboratories de Marcoussis Division Informatique Route de Nozay 91460 Marcoussis France	1
Vinay Apsingikar CMC Limited R & D Division 115 Sarojini Devi Road Secunderabad 500003 India	1

John Backus
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099 1

Jospeh H. Fasel
Los Alamos National Laboratory
C-10, MS B296
Los Alamos, NM 87545 1

Robert Floyd
Computer Science Department
Stanford University
Stanford, CA 94305 1

Joseph A. Goguen
SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, CA 94025 1

Peter Henderson
Department of Computer Science
SUNY at Stony Brook
Long Island, NY 11794 1

Paul Hudak
Yale University
Department of Computer Science
Box 2158, Yale Station
New Haven, CT 06520 1

Bharat Jayaraman
University of North Carolina
Department of Computer Science
New West Hall 035 A
Chapel Hill, NC 27514 1

A. Dain Samples
Computer Science Division - EECS
University of California
Berkeley, CA 94720 1

Mayer Schwartz
Computer Research Laboratory
MS 50-662
Tektronix, Inc.
P. O. Box 500
Beaverton, OR 97077 1

Guy L. Steele
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142 1

Richard Taylor
INMOS Limited

Whitefriars
Lewins Mead
Bristol BS1 2NP
UK

1