Acta Informatica 6, 133—151 (1976)

Affine Relationships Among Variables of a Program*

Michael Karr

Received May 8, 1974

Summary. Several optimizations of programs can be performed when in certain regions of a program equality relationships hold between a linear combination of the variables of the program and a constant. This paper presents a practical approach to detecting these relationships by considering the problem from the viewpoint of linear algebra. Key to the practicality of this approach is an algorithm for the calculation of the "sum" of linear subspaces.

1. Problems in Need of a Solution

A class of problems in the optimization of programs centers around the need to relate affine combinations of program variables (linear combinations plus a constant). In the development of the IVTRAN compiler for the ILLIAC IV, for instance, much effort is spent in the "parallelizing" of nested DO loops. One requirement is that the upper and lower limits for a given loop be expressible as a function of the index variables of containing DO loops. In practice, these relationships are almost invariably affine (linear plus a constant) in the other index combinations. Because of FORTRAN syntax restrictions, expressions cannot appear as DO loop limits; this directly gives us the problem of the determination of whether, at a given point in a program, a variable is some affine combination of other variables.

Naturally, the above problem could be solved most of the time by an ad hoc analyzer built especially for the purpose of recognizing the common situations. However, a seemingly unrelated problem may be expressed in the same terms. A common optimization known as "reduction in operator strength" occurs in source code, because wise programmers often know that their particular compiler does not do the transformation for them. This optimization converts a sequence of "expensive" operations—multiplications of a loop index by a constant—into a series of "cheaper" operations—additions of the constant to a "co-index". On serial machines, this transformation is indeed an optimization; however, on a parallel machine, it is preferable to do n multiplications at the same time than to perform n successive additions. Thus, the transformation must be reversed. This problem may at first seem difficult, and the usual suggestion is to find a "heuristic" which will solve it most of the time. But observe that such a transformation always stems from some linear expression involving a loop index, and

* This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by U. S. Army Research Office—Durham under Contract No. DAHC04-70-C-0023.

hence always introduces some affine relationship between the variable to be incremented by the series of additions (the co-index) and the loop index. Again, a knowledge of affine relationships among variables would enable us to solve a problem.

The above problems arise from the standpoint of optimization by parallelizing. Even in more classical optimization problems, a knowledge of affine relationships is applicable. For instance, constant propagation may be viewed as the recognition of such facts as $I=1$ at some point in a program (i.e., that statement is true every time flow of control is at the point). Such a relationship is a special case of an affine relationship among variables. This suggests that we could do a more general form of constant recognition, that is, we could recognize when linear forms are constants (even though their constituents might not be). Further, if in the detection of affine relationships, we make full use of the content of decision nodes, the constant detection machinery will thereby have a natural way of using the content of decision nodes in the detection of constants. For example, if the flowblock beginning at 1ØØ is uniquely accessible from the statement IF (I .EQ. Ø) GO TO 1ØØ, then all occurrences of I in that flowblock may be replaced by Ø.

Another application is in the recognition of common subexpressions which are not identical formally, but are the same because of relationships among variables. If these relationships are affine, then we can detect otherwise overlooked common subexpressions.

The existence of the common thread running through the above problems suggests at least making an attempt to develop a general technique for analyzing affine relationships among program variables. This paper shows that a unified approach to these problems is indeed feasible.

2. Representation of Affine Relationships

2.1 Choices Involved

Suppose that the scalar variables of a program are denoted by V_i; let $\boldsymbol{V}=\langle V_1, \ldots, V_n\rangle$. Each variable has values in a field F, which in practice may be the rationals or the reals. From a mathematical point of view, to say that a given affine relationship holds among the values of the variables at a given point in the program is to say that for any execution of the program, when the flow of control reaches the given point, then the vector of values $\boldsymbol{V}$ always lies in some fixed affine subspace[1] of F^n. As the affine relations become more numerous, the subspace has smaller and smaller dimension. If the subspace has dimension 0, then $\boldsymbol{V}$ is a constant, with each V_i taking on the value of the i^{th} coordinate of the point. If the relations are contradictory, then the subspace is the null set, and may be thought of as having dimension $-\infty$. At the other end, if there are no affine relationships, then the corresponding subspace is the entire space F^n, and has dimension n.

If we wish to make use of affine relationships, we must have a scheme for the finite representation of these (possibly infinite) spaces. There are basically two

1 Intuitively, an affine subspace is a point, line, plane, etc., not necessarily passing through the origin. See [1], Chapter XII for details.

methods for representing affine subspaces. One way is to represent a subspace by a basis of its linear component, plus an offset vector. A second way is to represent the space as the kernel of an affine transformation[2] from F^n to F^m for appropriate m. Luckily, there is no real debate over which method to choose, when we consider the fact that in most programs, the number of affine relationships will be small—i.e., the subspace will be large. The first method of representation requires as many basis elements as the dimension of the subspace.

In the second method, the representation is inversely related to the size of the space. Suppose that there are m independent affine relationships. These can be represented by a $m \times n$ matrix A, and a vector $\boldsymbol{c}$ of length m, where the map is:

$$\langle A, \boldsymbol{c}\rangle : F^n \to F^m \tag{2.1}$$
$$\boldsymbol{V} \mapsto A\boldsymbol{V} - \boldsymbol{c}$$
$$\ker \langle A, \boldsymbol{c}\rangle \overset{\text{def}}{=} \{\boldsymbol{V} \mid A\boldsymbol{V} = \boldsymbol{c}, \text{ i.e., } A\boldsymbol{V} - \boldsymbol{c} = \boldsymbol{0}\}.$$

The kernel of this map has dimension $n - m$. Thus, if there are no relations at all, the matrix and vector are "empty"; the more relations there are, the more rows the matrix A must have, but then that much more useful information has been extracted from the program. There is one affine subspace which cannot be represented by a matrix-vector pair, namely the null set; it may be represented by a reserved symbol, say Φ.

2.2 Canonical Form

There is a further issue in the representation of affine relationships. Several different matrix-vector pairs may represent essentially the same set of relationships; viewed as maps as in (2.1), their kernel is the same. We shall see that it is important that identical predicates have identical matrix-vector pairs representing them. Elementary linear algebra fortunately provides us with a "canonical form" for this problem. The property we require of this form is:

(2.2) Let $\langle A_i, \boldsymbol{c}_i\rangle$, $i = 1, 2$ be two matrix-vector pairs in canonical form. Then:

$$\ker \langle A_1, \boldsymbol{c}_1\rangle = \ker \langle A_2, \boldsymbol{c}_2\rangle \Leftrightarrow A_1 = A_2 \quad \text{and} \quad \boldsymbol{c}_1 = \boldsymbol{c}_2.$$

The particular canonical form which is appropriate for our purposes is "normalized reduced row-echelon form". Our reference on this is [2], to which the reader is referred for proofs; we shall present the definitions, because they are useful later in this paper.

(2.3) The matrix A is in *row-echelon* form $\overset{\text{def}}{\Leftrightarrow}$

(a) Every row of A has at least one non-zero entry.

(b) For any row i_0, let j_0 be the first column with a non-zero entry of the row. Then for all $i > i_0$, $j \leqq j_0$, $A_{ij} = 0$.

Row operations, namely those which multiply a row by a non-zero scalar, or which add a row to another row, or which permute two rows, may be used to

2 An affine transformation maps affine subspaces to affine subspaces. Its kernel is the set of elements mapped to the origin, and is itself an affine subspace.

transform any matrix A so that it satisfies (2.3 b). When performing row operations on $\langle A, \boldsymbol{c}\rangle$, $\boldsymbol{c}$ is treated as the $n+1^{\text{st}}$ column; such operations leave the kernel of $\langle A, \boldsymbol{c}\rangle$ unchanged. The kernel is also unchanged if an entire row of zeros is deleted from $\langle A, \boldsymbol{c}\rangle$. If the i^{th} row of A is zero, but $c_i \neq 0$, then the kernel of $\langle A, \boldsymbol{c}\rangle$ is null, and the space would be represented by Φ. Hence, we can transform a pair $\langle A, \boldsymbol{c}\rangle$ until the matrix is in row-echelon form, or the kernel is discovered to be null. (Deletions of zero rows may result in an "empty" matrix-vector pair, whose kernel is all of F^n—this is different from Φ, which is none of F^n.)

(2.4) Let A be in row-echelon form. Then the column index of the first non-zero entry of row i is denoted ϱ_i. The columns whose indices are $\{\varrho_i\}_{i=1}^m$ are called the *dependent* columns. The rest are the *independent* columns.

Notice that if $i_1 > i_2$, then $\varrho_{i_1} > \varrho_{i_2}$. The motivation for the names of column groupings comes from the theory of the solution of simultaneous linear equations.

(2.5) The matrix A is in *reduced row-echelon* form $\overset{\text{def}}{\Leftrightarrow}$

(a) A is in row-echelon form, and

(b) If $i < i_0$ then $A_{i\varrho_{i_0}} = 0$.

Any matrix-vector pair satisfying (2.3) may be made to satisfy (2.5) simply by applying appropriate row operations. At this stage, there is no possibility that the kernel may be discovered to be null.

(2.6) The matrix A is said to be *normalized* $\overset{\text{def}}{\Leftrightarrow}$ the first non-zero element of each row of A is 1.

Clearly, any matrix may be normalized by the row operations of multiplications by suitable non-zero elements of F. In practice, it may be more convenient to use other normalization conditions; the above is the most convenient for purposes of this paper.

3. Deriving Valid Relationships

Following Floyd [3], we shall attach our particular form of assertions to the arcs of a program graph: on arc x, the statement of affine relationships will be represented by $\langle A_x, \boldsymbol{c}_x\rangle$. The subject of this paper will not be the algorithm for "pushing" assertions around the program graph. Wegbreit [4] points out that this problem can be considered independently of the nature of the particular assertions which one is pushing. In this paper, we shall follow the basic outlines of [4]. The termination and correctness of proofs of that paper apply to the specific case analyzed here.

For purposes of exposition, we shall view the program graph as having three types of nodes: assignment, decision, and merge. No computation is contained in merge nodes, which have only one exit arc, as do assignment nodes. The computation of the right-hand side of an assignment statement and of the expression in a decision node are assumed *not* to affect the values of any variables. Thus, all side-effect phenomena must be modeled as assignment statements.

The basic idea of Wegbreit's algorithm as applied to affine relationships is to start with some $\langle A, \boldsymbol{c}\rangle$ on the entry arc to the program, which represents what is known about the variables at the start of execution. This is typically the empty matrix-vector pair (representing that nothing is known), though if the variables are initialized to 0, then the pair $\langle I, \mathbf{0}\rangle$ is appropriate, where I is the identity matrix. All of the other arcs are initialized to Φ.

For each of the three different types of nodes, we shall describe a transformation which specifies affine space(s) for the output arc(s) of the node in terms of 1) the affine space(s) on the input arc(s) to the node, and 2) where relevant, the content of the node. The transformations all have the property that if the input assertions are all true (for every execution of the program which passes through the arc) then the output assertions are true. The algorithm of [4] consists of applying these transformations until all the assertions are "stabilized"—i.e., the application of a transformation at any node results in no change in the assertions on the output arcs.

The proof in [4] that this algorithm terminates is, for affine assertions, due to the "ascending chain condition" which holds for finite-dimensional affine spaces—namely, any properly ascending chain of sub-spaces:

$$U_1 \subset U_2 \subset \ldots$$

is finite. This is obvious, for each of the subspaces U_i must have a dimension of at least one greater than U_{i-1}. Since the dimension of any U_i is limited by n, such a sequence cannot continue indefinitely.

The rest of this paper deals with the transformations at each of the three different kinds of nodes, and with the details of the calculations involved in carrying out these transformations. In the examples, we shall use the algorithm described loosely above, in more detail in [4].

4. Node Functions

4.1 Decision Nodes

The basic form of decision nodes which interests us is shown in Fig. 1. Note that on the "yes" arc, we can claim that $\boldsymbol{B}\boldsymbol{V}_{\text{yes}} = d$, that is, $\boldsymbol{V}_{\text{yes}} \in \ker \langle \boldsymbol{B}, d\rangle$ (we view the vector $\boldsymbol{B}$ as a matrix of one row, and the scalar d as a vector of one element). Since the decision leaves the values of $\boldsymbol{V}_x$ unaffected, $\boldsymbol{V}_{\text{yes}} = \boldsymbol{V}_x$, and we also have $\boldsymbol{V}_{\text{yes}} \in \ker \langle A_x, \boldsymbol{c}_x\rangle$; hence $\boldsymbol{V}_{\text{yes}} \in \ker \langle \boldsymbol{B}, d\rangle \cap \ker \langle A_x, \boldsymbol{c}_x\rangle$. Since the intersection of any two affine spaces is itself an affine space, there will be some representation for this intersection—either a matrix-vector pair, or the special symbol for null. Using the traditional symbol for conjunction of predicates:

(4.1) Let $\langle A_i, \boldsymbol{c}_i\rangle\ i = 1, 2$ be two matrix-vector pairs in canonical form. We have:

$$\langle A_1, \boldsymbol{c}_1\rangle \wedge \langle A_2, \boldsymbol{c}_2\rangle \stackrel{\text{def}}{=} \text{the canonical representation of } \ker \langle A_1, \boldsymbol{c}_1\rangle \cap \ker \langle A_2, \boldsymbol{c}_2\rangle.$$

This conjunction may be calculated by placing one matrix beneath the other, and performing row operations until canonical form is achieved, or the null kernel is

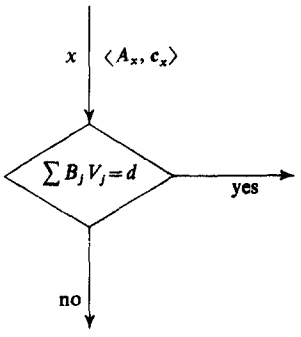


Fig. 1

detected. This gives us the appropriate predicate to be placed on the "yes" arc:

$$\langle A_x, \boldsymbol{c}_x\rangle \wedge \langle \boldsymbol{B}, d\rangle.$$

The predicate for the "no" arc cannot directly incorporate the affine non-equality. If $\langle A_x, \boldsymbol{c}_x\rangle \wedge \langle \boldsymbol{B}, d\rangle = \langle A_x, \boldsymbol{c}_x\rangle$, then $\ker \langle \boldsymbol{B}, d\rangle \subseteq \ker \langle A_x, \boldsymbol{c}_x\rangle$ and flow always leaves through the "yes" arc; in this case Φ is a valid predicate for A_{no}. Otherwise, we cannot add any information to $\langle A_x, \boldsymbol{c}_x\rangle$, and must be content with that predicate itself on the "no" arc. In summary, the predicate for the "no" arc is:

Φ if $\langle A_x, \boldsymbol{c}_x\rangle$ is the predicate for the "yes" arc;

$\langle A_x, \boldsymbol{c}_x\rangle$ otherwise.

In [5], a general study is made of how best to handle decision nodes which are not of the simple form of Fig. 1. For purposes of this paper, we may handle such nodes merely by placing $\langle A_x, \boldsymbol{c}_x\rangle$ on each of the output arcs. This is certainly valid, but it may not be as much information as could be gathered.

4.2 *Assignment Nodes*

Consider the assignment node in Fig. 2. What we desire is the "strongest" affine predicate $\langle A_y, \boldsymbol{c}_y\rangle$ which is consistent with $\langle A_x, \boldsymbol{c}_x\rangle$, and the assignment's being made. We also are interested in whether there is more than one "strongest" affine predicate suitable for $\langle A_y, \boldsymbol{c}_y\rangle$ (hopefully there is not). The next two subsections consider the two major types of assignment statements—those in which the assignment provides an invertible relationship between the $\langle A_x, \boldsymbol{c}_x\rangle$ and $\langle A_y, \boldsymbol{c}_y\rangle$, and the case where the relationship is non-invertible.

4.2.1 Invertible Assignment Nodes

Suppose the assignment T of Fig. 2 is of the form

$$V_{j_0} \leftarrow \sum_j B_j V_j + d \qquad B_{j_0} \neq 0. \tag{4.2}$$

The fact that $B_{j_0} \neq 0$ allows us to carry over our knowledge of the previous value of V_{j_0} to the new value of V_{j_0}, enabling us to perform a "change of coordinates"

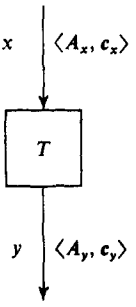


Fig. 2

on $\langle A_x, \mathbf{c}_x \rangle$. To see this, denote the values of the variables by V_{xj} on arc x, and V_{yj} on arc y. We start with:

$$V_{yj} = V_{xj} \quad \text{for } j \neq j_0,$$
$$V_{yj_0} = \sum_j B_j V_{xj} + d.$$

This is an affine transformation sending $\boldsymbol{V}_x$ to $\boldsymbol{V}_y$, which, as a short omitted calculation shows, may be inverted:

$$\begin{aligned} V_{xj} &= V_{yj} \quad j \neq j_0, \\ V_{xj_0} &= \Big(V_{yj_0} - \sum_{j \neq j_0} B_j V_{yj} - d\Big)/B_{j_0}. \end{aligned} \tag{4.3}$$

This expression for $\boldsymbol{V}_x$ in terms of $\boldsymbol{V}_y$ may be substituted into $A_x \boldsymbol{V}_x = c_x$ (which is, in fact, the assertion on the arc x). We obtain:

$$\forall i \left[\sum_{j \neq j_0} (A_{xij} - (A_{xij_0}/B_{j_0})\, B_j)\, V_{yj} + (A_{xij_0}/B_{j_0})\, V_{yj_0} = c_{xi} + (A_{xij_0}/B_{j_0})\, d \right]. \tag{4.4}$$

Picking out terms in the above expression, we let

$$\begin{aligned} A'_{yij} &\overset{\text{def}}{=} A_{xij} - (A_{xij_0}/B_{j_0})\, B_j \quad && \text{for } j \neq j_0, \quad \text{all } i \\ &\overset{\text{def}}{=} A_{xij_0}/B_{j_0} && j = j_0, \quad \text{all } i \\ c'_{yi} &\overset{\text{def}}{=} c_{xi} + (A_{xij_0}/B_{j_0})\, d && \text{for all } i, \end{aligned} \tag{4.5}$$

so that the conditions of (4.4) are equivalent to $A'_y \boldsymbol{V}_y = \mathbf{c}'_y$. We finally obtain $\langle A_y, \mathbf{c}_y \rangle$, the predicate for the arc y, by putting $\langle A'_y, \mathbf{c}'_y \rangle$ into canonical form.

A commonly occurring assignment statement is:

$$V_{j_0} = V_{j_0} + d. \tag{4.6}$$

In this case $B_{j_0} = 1$ and $B_j = 0$ for $j \neq j_0$. Substituting this into (4.5) gives:

$$\begin{aligned} A_{yij} &= A'_{yij} = A_{xij} \quad \text{for all } i, j \text{ (no row reductions are necessary)} \\ c_{yi} &= c'_{yi} \;\; = c_{xi} + A_{xij_0}\, d. \end{aligned} \tag{4.7}$$

Furthermore, if $d=0$, we have $\boldsymbol{c}_y=\boldsymbol{c}_x$, which is reassuring: if the assignment changes nothing, the affine space carries over unchanged.

4.2.2 Non-Invertible Assignment Nodes

If the assignment statement is not of the form of (4.1), then it appears that we cannot solve for $\boldsymbol{V}_x$ in terms of $\boldsymbol{V}_y$, and we have "lost" some information by the assignment to V_{j_0}. This loss has to do with the fact that old relationships involving V_{j_0} and other variables no longer hold true. If the j_0^{th} column of A_x is entirely zero, then $\langle A_x, \boldsymbol{c}_x\rangle$ is independent of the value of V_{j_0}, and is thus valid for the arc y. Otherwise, we must remove the dependence of $\langle A_x, \boldsymbol{c}_x\rangle$ on V_{j_0}. The obvious way to do this is to remove the rows of A_x which have a non-zero entry in the j_0^{th} column (and the corresponding elements of $\boldsymbol{c}_x$). A better way is to:

(4.8) Let i_0 be the *last* row in which the j_0^{th} entry is non-zero. Reduce rows $1, \ldots, i_0-1$ by row i_0 to make the j_0^{th} entry zero (rows $i_0+1, \ldots, m$ already have zeros as the j_0^{th} entry, by canonical form). Then, remove row i_0 and call the result $\langle A_y, \boldsymbol{c}_y\rangle$.

The reader may verify that the $\langle A_y, \boldsymbol{c}_y\rangle$ thus produced is in canonical form, and that its j_0^{th} column is zero. Nevertheless, the rule seems arbitrary, since it appears that the resulting space might be different if we had picked another row to reduce by, and then to eliminate; it might also depend upon column order (which affects canonical form). In fact, these fears are groundless, and (4.8) may be shown to be, in some sense, the unique "best" that can be done under the circumstances. This analysis, which also shows an underlying connection between the invertible and non-invertible case, is unfortunately too lengthy and general for inclusion here; the reader is referred to [5].

Once we have weakened the predicate, if necessary, to delete information about V_{j_0}, we again examine the right-hand side of the assignment. If it is some non-affine function of the other program variables, then we may make no new statements of affine relationship. If, however, the statement is of the form:

$$V_{j_0} \leftarrow \sum_{j \neq j_0} B_j V_j + d. \tag{4.9}$$

then we may add to our list of other relationships the statement

$$\sum_j B'_j V_j = d \quad \text{where} \quad B'_j = \begin{cases} -B_j & \text{if } j \neq j_0 \\ 1 & \text{otherwise} \end{cases}.$$

Applying (4.1), the better definition for this case is:

$$\langle A_y, \boldsymbol{c}_y\rangle \overset{\text{def}}{=} \langle \boldsymbol{B}', \boldsymbol{d}\rangle \wedge \langle A'_x, \boldsymbol{c}'_x\rangle \tag{4.10}$$

where $\langle A'_x, \boldsymbol{c}'_x\rangle$ is the matrix-vector pair obtained in (4.8).

4.3 Merge Nodes

It is in this section that the most difficult problem is confronted. Consider Fig. 3. On each of the input arcs x, we have some affine space $\langle A_x, \boldsymbol{c}_x\rangle$. What is

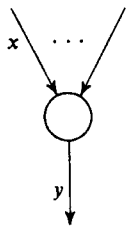


Fig. 3

desired on the output arc y is the strongest *affine* statement which can be made on the basis of the information on the input arcs. If the setwise union of affine spaces were itself affine, the solution would be immediate. However, this is not at all the case, and we must do something weaker. The next best thing which we can do is to define the "affine union" of affine spaces to be the "smallest" affine space which contains the set-wise union.

(4.11) Let U_1, U_2 be affine spaces contained in F^n. We say that an affine space W is their *affine union* $\overset{\text{def}}{\Leftrightarrow}$

(a) $U_1, U_2 \subseteq W$ (W contains the setwise union)

(b) If $U_1, U_2 \subseteq W'$ then $W \subseteq W'$ (W is the smallest space satisfying (a)).

By general lattice theory, W is necessarily unique (see [6], p. 6). Its existence is guaranteed ([6], p. 7, Theorem 6) using only the fact that the intersection of any number of affine subspaces is an affine subspace. This allows the following definition, this time using the traditional symbol for disjunction.

(4.12) Let $\langle A_i, \boldsymbol{c}_i \rangle$ be two matrix-vector pairs in canonical form. The *affine disjunction* of these two pairs is:

$$\langle A_1, \boldsymbol{c}_1 \rangle \vee \langle A_2, \boldsymbol{c}_2 \rangle \overset{\text{def}}{=} \text{the canonical representation of the affine union of ker } \langle A_1, \boldsymbol{c}_1 \rangle \text{ and ker } \langle A_2, \boldsymbol{c}_2 \rangle.$$

The problem of merge nodes has been confronted, but only partly solved. What we must have, of course, is an algorithm for efficiently *computing* the canonical representation.

5. Calculation of Affine Disjunction

5.1 Reduction to Linear Disjunction

As we saw in Section 4.1, the representation of the intersection of two affine spaces may be easily calculated from the representation of each of the intersectands, by merely concatenating the two matrices involved, and using row operations to get canonical form. The problem of calculating the affine union is somewhat more difficult. We start by reducing it to a simpler algebraic object.

We must first introduce a concept from elementary linear algebra.

(5.1) A *linear* subspace of F^n is an affine subspace which contains the origin. (This is hardly the traditional definition, but is convenient here.)

We may represent a linear subspace by a matrix, similarly to the way in which a matrix-vector pair represents an affine subspace.

(5.2) For an $m \times n$ matrix A, view

$$A: F^n \to F^m$$

$$\boldsymbol{V} \mapsto A\boldsymbol{V}.$$

$$\ker A \stackrel{\text{def}}{=} \{\boldsymbol{V} | A\boldsymbol{V} = 0\}.$$

Analogous to the situation in 2.2, there is a one-one correspondence between linear subspaces of F^n, and matrices in reduced row echelon form.

Observe that the affine union of two linear subspaces is again linear; this is traditionally called the *sum* of the two subspaces, and is denoted by $+$. The correspondence between linear subspaces and matrices in canonical form thus induces an operation on these matrices, analogous to (4.12). We shall again use $\vee$ to denote this operation; context will distinguish this "linear disjunction" from the "affine disjunction" which previously used the same symbol. The two operations are in fact closely related. To see how, let

(5.3) $(A, \boldsymbol{c}) \stackrel{\text{def}}{=}$ the $m \times (n+1)$ matrix formed from the $m \times n$ matrix A by adjoining the m-element vector $\boldsymbol{c}$ as the $n+1^{\text{st}}$ column.

Quite conveniently, linear sum in F^{n+1} corresponds to affine union in F^n:

(5.4) Let A_1, A_2 be n-column matrices in canonical form, and $\boldsymbol{c}_1, \boldsymbol{c}_2$ be vectors of the proper lengths. Let $\langle A, \boldsymbol{c}\rangle$ be the matrix-vector pair defined by $\langle A_1, \boldsymbol{c}_1\rangle \vee \langle A_2, \boldsymbol{c}_2\rangle$. Then

$$(A, \boldsymbol{c}) = (A_1, \boldsymbol{c}_1) \vee (A_2, \boldsymbol{c}_2).$$

In other words, the $n+1$-column matrix produced by linear disjunction, and that produced by affine disjunction, are the same.

(A proof may be found in the Appendix.) This reduces the problem of affine disjunction to that of linear disjunction, which we consider next.

5.2 Calculation of the Linear Disjunction

We shall give here an efficient algorithm, in terms of both time and space, for the calculation of the matrix which represents the union of two linear subspaces, given the two matrices which represent the subspaces whose union is to be taken.

We shall assume that the matrices A and B are given in canonical form, and we shall produce a matrix C, again in canonical form, which represents the union. The algorithm consists of n steps, where n is the number of columns. It proceeds by modifying the original natrices A and B so that at the end of step s, the first s columns of the modified A and B are the same. At the end of n steps, both matrices

are the same; the affine union of identical spaces is of course the space itself, and so we will have arrived at the desired result. The algorithm is started by:

(5.5) $A^{(0)} \stackrel{\text{def}}{=} A_0^{(0)} \stackrel{\text{def}}{=} A$; $B^{(0)} \stackrel{\text{def}}{=} B_0^{(0)} \stackrel{\text{def}}{=} B$;
$C^{(0)}$ is the "matrix with no rows or columns".
For any matrix D, $|D| \stackrel{\text{def}}{=}$ the number of rows of D.

At the completion of step s, the following four conditions will hold

(5.6.1) $\ker A^{(s)} + \ker B^{(s)} = \ker A + \ker B$ (sum of linear subspaces)

(5.6.2) $$A^{(s)} = \left(\begin{array}{c|c} C^{(s)} & \\ \hline & A_0^{(s)} \end{array}\right); \qquad B^{(s)} = \left(\begin{array}{c|c} C^{(s)} & \\ \hline & B_0^{(s)} \end{array}\right).$$

(Unlabelled submatrices of a partition are assumed to be filled with zeros. $A_0^{(s)}$, $B_0^{(s)}$ are defined to be submatrices of $A^{(s)}$, $B^{(s)}$ respectively, consisting of of columns $s+1, \ldots, n$.)

(5.6.3) $A^{(s)}$, $B^{(s)}$ and $C^{(s)}$ are all in canonical form

(5.6.4) $C^{(s)}$ is s columns wide. (We will also have $0 \leqq |C^{(s)}| \leqq s$, where the fact that it has no rows but s columns means that the first s columns of $A^{(s)}$ and $B^{(s)}$ are zero.)

There are essentially three cases which apply at each step, depending upon the values on $A^{(s-1)}$ and $B^{(s-1)}$ of the element down one and over one from the $C^{(s-1)}$ submatrix. Assume that we have completed $s-1$ steps of the algorithm, and are starting step s. For convenience, let $r \stackrel{\text{def}}{=} |C^{(s-1)}| + 1$.

Case 1: $A_{rs}^{(s-1)} = B_{rs}^{(s-1)} = 1$

$$A^{(s-1)} = \left(\begin{array}{c|c|c} & \vdots & \\ C^{(s-1)} & 0 & A_0^{(s)} \\ \hline & 1 & \end{array}\right); \qquad B^{(s-1)} = \left(\begin{array}{c|c|c} & \vdots & \\ C^{(s-1)} & 0 & B_0^{(s)} \\ \hline & 1 & \end{array}\right).$$

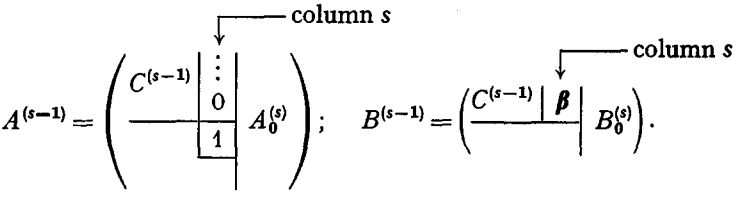
(column s marked above the middle column)

The zero elements of columns s in each matrix are forced because of the reduced row-echelon form. We can satisfy the induction requirements by defining

$$A^{(s)} \stackrel{\text{def}}{=} A^{(s-1)}; \qquad B^{(s)} \stackrel{\text{def}}{=} B^{(s-1)}; \qquad C^{(s)} \stackrel{\text{def}}{=} \left(\begin{array}{c|c} C^{(s-1)} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \ldots 0 & 1 \end{array}\right).$$

In other words, we adjoin a row and a column to the $C^{(s-1)}$ matrix, and without making any modifications to the matrices, obtain the conditions for step s.

Case 2: $A_{rs}^{(s-1)}=1$, $B_{rs}^{(s-1)}=0$ (Symmetrical if 0 and 1 are exchanged)



$$A^{(s-1)}=\left(\begin{array}{c|c|c} C^{(s-1)} & \begin{matrix}\vdots\\ 0\end{matrix} & \\ \hline & 1 & A_0^{(s)} \end{array}\right); \quad B^{(s-1)}=\left(\begin{array}{c|c|c} C^{(s-1)} & \boldsymbol{\beta} & \\ \hline & & B_0^{(s)} \end{array}\right).$$

This also covers the case in which $|C^{(s-1)}|=|B^{(s-1)}|$, i.e. when r is greater than the number of rows of $B^{(s-1)}$. We shall modify the matrix $A^{(s-1)}$ to obtain $A^{(s)}$. The first step in the modification is to obtain $\boldsymbol{\beta}$ in the first $r-1$ positions of column s in the A matrix. To do this, for any row i, $1\leqq i\leqq r-1$, we multiply row r of $A^{(s-1)}$ by β_i, and add it to row i of $A^{(s-1)}$. This operation leaves the $C^{(s-1)}$ submatrix of $A^{(s-1)}$ undisturbed, but it may well change the upper rows of $A_0^{(s-1)}$. Having performed this set of row operations, delete row r from $A^{(s-1)}$. The matrix thus obtained is defined to be $A^{(s)}$. We also let:

$$B^{(s)} \stackrel{\text{def}}{=} B^{(s-1)} \quad \text{and} \quad C^{(s)} \stackrel{\text{def}}{=} (C^{(s-1)}|\boldsymbol{\beta}).$$

The fact this operation maintains the validity of (5.6.1) is proved in the Appendix. The reader may verify that the conditions (5.6.2)—(5.6.4) are maintained by the definitions of $A^{(s)}$, $B^{(s)}$ and $C^{(s)}$ from $A^{(s-1)}$, $B^{(s-1)}$, $C^{(s-1)}$.

Case 3: $A_{rs}^{(s-1)}=B_{rs}^{(s-1)}=0$

$$A^{(s-1)}=\left(\begin{array}{c|c|c} C^{(s-1)} & \boldsymbol{\alpha} & \\ \hline & & A_0^{(s)} \end{array}\right); \quad B^{(s-1)}=\left(\begin{array}{c|c|c} C^{(s-1)} & \boldsymbol{\beta} & \\ \hline & & B_0^{(s)} \end{array}\right).$$

This also covers the case when either or both $|C^{(s-1)}|=|A^{(s-1)}|$, $|C^{(s-1)}|=|B^{(s-1)}|$. If $\boldsymbol{\alpha}=\boldsymbol{\beta}$, this step is immediate. Let:

$$C^{(s)} \stackrel{\text{def}}{=} (C^{(s-1)}|\boldsymbol{\alpha}), \quad A^{(s)} \stackrel{\text{def}}{=} A^{(s-1)}, \quad B^{(s)} \stackrel{\text{def}}{=} B^{(s-1)},$$

and induction assumptions of (5.6) are all immediate. If $\boldsymbol{\alpha}\neq\boldsymbol{\beta}$, we shall be interested in the row t, $1\leqq t<r$, where t is the largest number such that $\alpha_t\neq\beta_t$. We shall eventually delete row t from both $A^{(s-1)}$ and $B^{(s-1)}$, but before doing so, shall perform row operations in each matrix so that after the deletion of row t, the first s columns will be identical, and it will be possible to define $C^{(s)}$. In each of $A^{(s-1)}$, $B^{(s-1)}$ for all rows i, $1\leqq i<t$, multiply row t of the matrix by $(\alpha_i-\beta_i)/(\alpha_t-\beta_t)$, and subtract from row i. Having done this, delete row t from each matrix. Define the results to be $A^{(s)}$ and $B^{(s)}$, respectively.

We claim that the first s columns of the two matrices are identical. The proof of this for the first $s-1$ columns follows from the fact that those columns were the same in $A^{(s-1)}$ and $B^{(s-1)}$, and underwent identical row operations. For column s, we need be concerned only with the elements $A_{is}^{(s)}$, $B_{is}^{(s)}$, for $1\leqq i<t$, since the other elements are the same by the definition of t, the deletion of row t from each matrix, and the properties of row-echelon form. For the remaining elements, by

virtue of the row operations we performed, we have:

$$A_{is}^{(s)} = \alpha_i - \left(\frac{\alpha_i - \beta_i}{\alpha_t - \beta_t}\right)\alpha_t = \frac{-\alpha_i\beta_t + \beta_i\alpha_t}{\alpha_t - \beta_t} = \beta_i - \left(\frac{\alpha_i - \beta_i}{\alpha_t - \beta_t}\right)\beta_t = B_{is}^{(s)}.$$

Thus, we can define $C^{(s)}$ to be the submatrix of either $A^{(s)}$ or $B^{(s)}$ consisting of the first s columns of the first $r-2$ rows.

(It is in this case that we may arrive at a matrix $C^{(s)}$ with no rows, but s columns. See (5.6.4).)

Given the above definitions of $A^{(s)}$, $B^{(s)}$, and $C^{(s)}$, the continued correctness of (5.6.1) will be shown in the Appendix, and (5.6.2)—(5.6.4) are left to the reader.

6. Some Examples

6.1 Pathological Flow Properties

Consider the program graph in Fig. 4. It is not our current concern here that we have an infinite loop; rather, that on arc ① we have $M-N=1$ and on arc ② we have $N-M=1$, which is clear to a human after a relatively short inspection. The point is not the subtlety of the affine relationship mechanism, which is trivial here. This example shows off the power of Wegbreit's Algorithm which works on quite messy graphs—note that this graph is "irreducible", as the term is used in interval analysis. However, Wegbreit's Algorithm "notices" the consistent loop-boundary conditions, and generates a non-trivial assertion on each arc of the loop.

We start at the entry arc, and push the trivial subspace down until we hit the two initialization nodes. This results in the following assertions attached to their respective arcs:

$$③\ M-N=1, \quad ④\ M-N=-1.$$

At the merge nodes [c] and [d], the null space is on one each of the incoming arcs, so the above relationships pass through to arcs ① and ②, respectively.

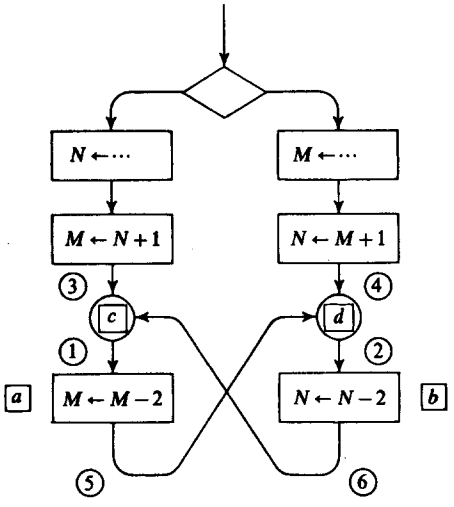


Fig. 4

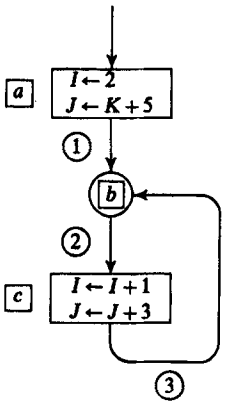


Fig. 5

Using the formula (4.7), we get:

$$\text{from node } \boxed{a}\text{:} \quad c_{⑤} = 1 + 1 \cdot (-2) = -1,$$

$$\text{from node } \boxed{b}\text{:} \quad c_{⑥} = -1 + (-1)\,(-2) = 1.$$

Thus we get assertions

$$⑤\ M - N = 1, \qquad ⑥\ M - N = 1.$$

Now, we are at the point where we must perform affine unions at $\boxed{c}$ and $\boxed{d}$.

$$\boxed{c}\text{: } \langle (1, -1), 1\rangle \cup \langle (1, -1), 1\rangle \qquad \boxed{d}\text{: } \langle (1, -1), -1\rangle \cup \langle (1, -1), -1\rangle.$$

It is obvious what we get as unions here, so we postpone an example of the computation for a more enlightening case. Note that the unions specify the affine assertions which are already on arcs ① and ③ respectively, so that the algorithm terminates.

6.2 Increasing Operator Strength

We now give an example of one of the applications mentioned at the beginning of this paper. In this case, the flow structure is quite simple, and the affine relationship analysis somewhat more subtle. Fig. 5 shows a simple loop with two counters, being incremented by 1 and 3, respectively, each time through the loop. Several assignments have been grouped into a single node, for the sake of conciseness. At the start we obtain:

$$①\text{: } \begin{matrix} I & = 2 \\ J - K & = 5 \end{matrix} \quad \text{or equivalently} \quad \left\langle \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}, \begin{pmatrix} 2 \\ 5 \end{pmatrix} \right\rangle.$$

This also pushes through to ②. At node $\boxed{c}$, we apply (4.7) twice to obtain:

$$③\text{: } \left\langle \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}, \begin{pmatrix} 3 \\ 8 \end{pmatrix} \right\rangle.$$

Then, we are back at node [b], where we must perform the disjunction of the predicates on arcs ① and ③, i.e.

$$\begin{pmatrix}1 & 0 & 0 & 2\\ 0 & 1 & -1 & 5\end{pmatrix} \vee \begin{pmatrix}1 & 0 & 0 & 3\\ 0 & 1 & -1 & 8\end{pmatrix}.$$

We now run through the algorithm of Section 5.4. Two quick applications of case 1, followed by the trivial application of case 3 leave $A^{(3)}$ and $B^{(3)}$ equal to the original A and B, respectively. We have

$$C^{(3)} = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & -1\end{pmatrix}.$$

At column four, we have a non-trivial case 3. In the notation of that section,

$$\alpha = (2, 5) \quad \text{and} \quad \beta = (3, 8).$$

The last row on which α and β differ is given by $t = 2$. Observe:

$$(\alpha_1 - \beta_1)/(\alpha_2 - \beta_2) = (2-3)/(5-8) = 1/3.$$

In each of the matrices, we multiply the second row by 1/3, and subtract it from the first row. We then delete the second row:

$$(1 \quad -1/3 \quad 1/3 \quad 2-5/3) \vee (1 \quad -1/3 \quad 1/3 \quad 3-8/3) = (1 \quad -1/3 \quad 1/3 \quad 1/3).$$

Thus the assertion on ② is:

$$②: \langle (1 \quad -1/3 \quad 1/3), 1/3 \rangle \quad \text{or equivalently } 3I - J + K = 1.$$

Now we consider the effect of the assignment node at [c]. For clarity, we now consider the assignments one at a time. After $I \leftarrow I + 1$, we have:

$$c = 1/3 + 1 \cdot 1 = 4/3, \quad A \text{ is unchanged.}$$

However, after $J \leftarrow J + 3$, we have:

$$c = 4/3 + (-1/3) \cdot 3 = 1/3, \quad A \text{ is unchanged.}$$

In other words, the two assignments have a combined effect of nil on the assertion, and we also obtain:

$$③: \langle (1 \quad -1/3 \quad 1/3), 1/3 \rangle.$$

We are at node [b] a third time, and must calculate the linear disjunction:

$$(1 \quad -1/3 \quad 1/3 \quad 1/3) \vee \begin{pmatrix}1 & 0 & 0 & 2\\ 0 & 1 & -1 & 5\end{pmatrix}.$$

For $s = 1$, we apply case 1, but with $s = 2$, we must use case 2. This requires that in the second argument, we subtract one third times the second row from the first, and delete the second. This yields:

$$(1 \quad -1/3 \quad 1/3 \quad 2-5/3) = (1 \quad -1/3 \quad 1/3 \quad 1/3).$$

For $s=3$ and 4 we have trivial applications of case 3, so that we again wind up with the same result for ②, and may terminate the algorithm.

Hence, except for the undrawn arc internal to node ⓒ, inside the loop have the relationship:

$$3I-J+K=1, \quad \text{i.e.,} \qquad J=K+3I-1.$$

This has been done without having to intelligently hypothesize an invariant relationship; affine disjunction takes care of this.

6.3 Constant Detection

Given the relationships $\langle A, \boldsymbol{c}\rangle$ on a given arc, in order to determine whether a linear form $\boldsymbol{B}\cdot\boldsymbol{V}$ is a constant, merely attempt to express $\boldsymbol{B}$ as a linear combination of the rows of A. If this is not possible, then we cannot conclude, at least on the basis of $\boldsymbol{V}\in\ker\langle A, \boldsymbol{c}\rangle$, that $\boldsymbol{B}\cdot\boldsymbol{V}$ is a constant. This is a direct consequence of the theory of the general solution of simulataneous equations, which says that under this linear independence, the system

$$\left\langle \begin{pmatrix} A \\ \boldsymbol{B} \end{pmatrix}, \begin{pmatrix} \boldsymbol{c} \\ 0 \end{pmatrix} \right\rangle \tag{6.1}$$

has fewer solutions than $\langle A, \boldsymbol{c}\rangle$, i.e. there are two solutions V_1, V_2 for $\langle A, \boldsymbol{c}\rangle$ which differ: $\boldsymbol{B}V_1 \neq \boldsymbol{B}V_2$, so that at most one is a solution to (6.1). On the other hand, if

$$\boldsymbol{B}=\sum_{i=1}^{m}\lambda_i\boldsymbol{A}_i \qquad (m=\text{number of rows of } A),$$

then

$$\boldsymbol{B}\cdot\boldsymbol{V}=\left(\sum_{i=1}^{m}\lambda_i\boldsymbol{A}_i\right)\cdot\boldsymbol{V}=\sum_{i=1}^{m}\lambda_i(\boldsymbol{A}_i\cdot\boldsymbol{V})=\sum_{i=1}^{m}\lambda_i c_i,$$

so that $\boldsymbol{B}\cdot\boldsymbol{V}$ is an obvious constant.

Since A is in row-echelon form, trying to find the desired linear combination is trivial: merely append $\langle \boldsymbol{B}, 0\rangle$ to $\langle A, \boldsymbol{c}\rangle$ and do row reductions. If the last row goes to zero in the first n entries, the negative of the desired constant,

$$-\sum_{i=1}^{n}\lambda_i c_{xi},$$

appears in the $n+1^{\text{st}}$ column; otherwise, a new dependent column emerges, and $\boldsymbol{B}$ is linearly independent of the rows of A. In particular, a variable, V_j, is a constant $\Leftrightarrow j$ is a dependent variable of A, and the (unique) row i in which column j is non-zero ($i=\varrho^{-1}(j)$) is 0 everywhere except the j^{th} position. The constant is given in the special case by:

$$c_i/A_{ij}.$$

To illustrate the power of this approach, we present the contrived but showy example of Fig. 6.

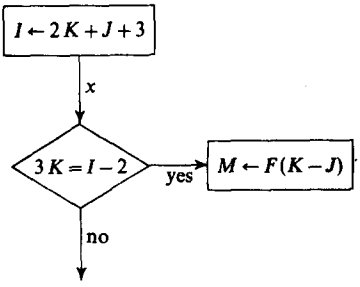


Fig. 6

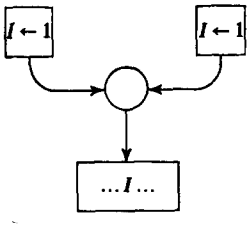


Fig. 7

Letting $\boldsymbol{V}=(I, J, K)$, on arc x we have $\langle(1\ -1\ -2), 3\rangle$. Using the condition in the decision node, we calculate the $\langle A_{\text{yes}}, \boldsymbol{c}_{\text{yes}}\rangle$ by:

$$\langle(1\ -1\ -2), 3\rangle \wedge \langle(1\ \ 0\ -3), 2\rangle = \left\langle \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right\rangle.$$

To determine the linear dependence, if any, of $K-J$ on the rows of this matrix, we adjoin the row $(0\ -1\ 1\ 0)$ and reduce:

$$\begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \text{ gets reduced to } \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

Thus, $K-J$ reduces to the constant $-(-1)=1$, as can be verified directly. This example shows both advantages of "affine constant detection" (constancy of an affine expression, knowledge gleaned from a decision node). In real world situations, such a situation is unlikely; however, the occurrence of an example which uses one advantage or the other is comparatively common.

An additional bonus provided by this method is the natural way in which constants are detected through merges. For example, see Fig. 7. The fact that the use of I is discovered to be a constant shows the advantage of "constant detection" over the more classical "constant propagation". While the latter method could be fixed up to handle such cases as this, such an approach would be an ad hoc addition to the basic method, and not a natural consequence of it.

Appendix

Proof of (5.4) Let $(A_0, \boldsymbol{c}_0) \stackrel{\text{def}}{=} (A_1, \boldsymbol{c}_1) \vee (A_2, \boldsymbol{c}_2)$. To show that $(A_0, \boldsymbol{c}_0) = (A, \boldsymbol{c})$, we merely show that the kernels are equal. For any B, $\boldsymbol{d}$, and $\boldsymbol{V} \in \ker (B, \boldsymbol{d})$:

$$\ker (B, \boldsymbol{d}) = \{\lambda \langle \boldsymbol{V}, -1 \rangle + \boldsymbol{W} | \boldsymbol{W} \in \ker B, \lambda \in F\}. \tag{1}$$

To prove this, note that $\ker (B, \boldsymbol{d})$ is at least this big, and that the set is closed under addition and under multiplication from F.

Pick any $\boldsymbol{V}_i \in \ker \langle A_i, \boldsymbol{c}_i \rangle$, $i = 1, 2$. Then $\omega_1 \boldsymbol{V}_1 + \omega_2 \boldsymbol{V}_2 \in \ker \langle A, \boldsymbol{c} \rangle$, because affine spaces (in particular, the union) are closed under weighted average ([1], p. 422). Generalizing the above formula slightly, we observe that:

$$\ker (A, \boldsymbol{c}) = \{\lambda \langle \omega_1 \boldsymbol{V}_1 + \omega_2 \boldsymbol{V}_2, -1 \rangle + \boldsymbol{W}_1 + \boldsymbol{W}_2 | \boldsymbol{W}_i \in \ker A_i, \lambda \in F\}. \tag{2}$$

Again, the kernel may be seen to be at least this big; and the above set is a linear subspace. But since $-1 = -\omega_1 - \omega_2$, we have:

$$\begin{aligned} \ker (A, \boldsymbol{c}) &= \left\{ \sum_{i=1}^{2} [\lambda \omega_i \langle \boldsymbol{V}_i, -1 \rangle + \boldsymbol{W}_i] \,|\, \boldsymbol{W}_i \in \ker A_i, \lambda \in F \right\} \\ &= \{X_1 + X_2 | X_i \in \ker (A_i, \boldsymbol{c}_i)\} \\ &= \ker (A_0, \boldsymbol{c}_0). \end{aligned}$$

The first equality is mere rearrangement of (2), the second uses (1), and the third is a property of the sum of linear subspaces ([1], p. 198). The identity of the kernels completes the proof.

Validity of (5.6.1) for Cases 2 and 3

We first observe that removing rows from the matrices enlarges the spaces, so that we immediately see that

$$\ker A^{(s-1)} + \ker B^{(s-1)} \leq \ker A^{(s)} + \ker B^{(s)}$$

If we can prove equality here, induction will immediately give us (5.6.1). Equality may be proved by showing that the dimensions are equal, which in turn may be shown by proving that the ranks of the matrices are equal. Using (5.6.3), we must show:

$$|A^{(s-1)} \vee B^{(s-1)}| = |A^{(s)} \vee B^{(s)}|$$

In case 2, view the deleted row as a one row matrix D, and observe that:

$$\begin{aligned} A^{(s-1)} &= A^{(s)} \wedge D, \\ |A^{(s)} \wedge D| &= |A^{(s)}| + 1. \end{aligned} \tag{*}$$

Notice that column s will be an independent column of $A^{(s)} \wedge B^{(s)}$; since D has a 1 in this column and 0 in previous columns,

$$|A^{(s)} \wedge B^{(s)} \wedge D| = |A^{(s)} \wedge B^{(s)}| + 1. \tag{†}$$

Hence

$$\begin{aligned}
|A^{(s-1)} \vee B^{(s-1)}| &= |A^{(s-1)} \vee B^{(s)}| && \text{(because } B^{(s)} = B^{(s-1)}\text{)} \\
&= |A^{(s-1)}| + |B^{(s)}| - |A^{(s)} \wedge D \wedge B^{(s)}| && \text{(dimension formula, (*))} \\
&= |A^{(s)}| + 1 + |B^{(s)}| - (|A^{(s)} \wedge B^{(s)}| + 1) && \text{(by (*), (†))} \\
&= |A^{(s)} \vee B^{(s)}| && \text{(cancellation of ones, dimension formula).}
\end{aligned}$$

(The required dimension formula may be found in [1], p. 235, Corollary 2.) This proves equality for case 2.

For case 3, the reasoning is quite similar. Let D be the row removed from $A^{(s-1)}$ and E be the row removed from $B^{(s-1)}$. In addition to (*), we have:

$$(**) \qquad \begin{aligned} B^{(s-1)} &= B^{(s)} \wedge E, \\ |B^{(s)} \wedge E| &= |B^{(s)}| + 1. \end{aligned}$$

Observe that both column s and column $\varrho^{-1}(t)$ (the position of the first element of this row in $C^{(s-1)}$) are independent columns of $A^{(s)}$ and $B^{(s)}$; also, that the two columns will be dependent columns of $D \wedge E$ (think about doing the row reductions necessary to put this in canonical form). We conclude that:

$$(††) \qquad |A^{(s)} \wedge B^{(s)} \wedge D \wedge E| = |A^{(s)} \wedge B^{(s)}| + |D \wedge E| = |A^{(s)} \wedge B^{(s)}| + 2.$$

Thus,

$$\begin{aligned}
|A^{(s-1)} \vee B^{(s-1)}| &= |(A^{(s)} \wedge D) \vee (B^{(s)} \wedge E)| && \text{(by (*), (**))} \\
&= |A^{(s)}| + 1 + |B^{(s)}| + 1 - (|A^{(s)} \wedge B^{(s)}| + 2) && \text{(dim. formula, (*), (**), (††))} \\
&= |A^{(s)} \vee B^{(s)}| && \text{(cancellation, dimension formula).}
\end{aligned}$$

This completes the proof that the algorithm for linear disjunction is correct.

References

1. MacLane, S., Birkhoff, G.: Algebra. New York: MacMillan 1967
2. Munkres, J. R.: Elementary linear algebra. Reading (Mass.): Addison-Wesley 1964
3. Floyd, R.: Assigning meanings to programs. In: Schwartz, J. (ed.): Mathematical aspects of computer science **19**. Providence (R.I.): American Mathematical Society 1967, p. 19–32
4. Wegbreit, Ben: Property extraction in well-Founded property sets. Center for Research in Computing Technology, Harvard University, Cambridge (Mass.) and Computer Science Division, Bolt, Beranek, and Newman, Inc., Cambridge (Mass.), February 1973
5. Karr, M.: Gathering information about Programs. Massachusetts Computer Associates, Inc., (In preparation)
6. Birkhoff, G.: Lattice theory. Colloquium Publication XXV, 3. Ed., Providence (R.I.): American Mathematical Society 1973

Michael Karr
Massachusetts Computer Associates Inc.
Lakeside Office Park
Wakefield, Mass. 01880
U.S.A.