Modern Dependence Testing

Carl D. Offner

September 24, 1996

Abstract

This purely expository paper was written as part of the HPF project at Digital. We needed a good dependence analyzer, and I wrote this both to provide a good description of what dependence analysis consists of and to see what the state of the art was at that time. I concluded that Pugh's Omega test was really the method of choice

The discussion in most places follows the original papers quite closely, although the exposition of many topics has been cleaned up quite a bit.

Contents

1 Introduction

We are given as input a loop nest and two array references in that nest. We want to determine

- if a dependence exists between those two references or not, and
- if such a dependence exists, the nature of the dependence; that is,
  1. whether it is a true, anti, or output;
  2. the direction and distance vectors.

Two matters we ignore in this initial report are:

1. scalar dependences. The techniques for this will be dealt with separately.
2. array aliasing. We assume that such aliasing has been handled so that the dependence analyzer sees arrays with the same name.

1.1 Restrictions

We are only going to consider dependence tests that consider affine subscript expressions. Any other form of subscript expression will be presumed to involve all possible dependences. In practice, this seems to be justified. The other kinds of subscript expressions that might occur are

- expressions involving non-linear terms in the loop variables, such as $i * i$ or $i * j$;
- expressions involving indirection arrays; i.e., vector-valued subscripts;
- expressions involving function calls.

It does not seem profitable to spend the large amount of compilation time that would be needed to analyze the first two of these kinds of expressions for dependence. In the vast majority of cases, a dependence can always be assumed to exist in either of these cases.

Function calls are a different matter. In many important cases they do not lead to dependences. Discovering this, however, requires some form of interprocedural analysis. Although this report does not concern itself with interprocedural issues, the techniques presented here become considerably more valuable when such interprocedural information is available, as has been strikingly demonstrated in the work of Monica Lam and her students at Stanford.

Interprocedural analysis is also important in another respect: in many cases, parameters such as loop bounds are passed in to a procedure. If interprocedural analysis can detect that these parameters are really compile-time constants, the dependence analyzer will execute much more quickly, and give much more useful results.

1.2 The Approach We Take

There are two approaches one might take to the problem of dependence testing:

1. Construct a sequence of simple *filter tests*. Each such test will recognize a small class of commonly occurring patterns. Any dependence problem encountered by the compiler for which there is no filter test provided will be assumed to have a dependence.
2. Construct a general method that can be used on any dependence problem.

The filter test approach (as exemplified in Goff, Kennedy, and Tseng's work at Rice) works well because most dependence problems really are pretty simple. Filter tests catch nearly all of them, and filter tests are easy to implement. The issues that have to be confronted when using this approach are:

- What do you do with the dependence problems that are missed by the filter tests?
- How do you organize the implementation so that it is robust and extensible?

The other technique—constructing a general method—is more in the spirit of modern compiler technology. Its advantage is that, if it is well implemented, it is pretty certain to give excellent results. Its disadvantage is that general techniques can be inefficient. For this reason, it is only recently that general techniques for dependence analysis have become popular, even in the world of academic research compilers.

In fact, these two methods as commonly implemented are not as completely opposed as I have indicated. People who implement a series of filter tests commonly also include a general purpose test at the end to try to mop up the tests that got away. And a general-purpose implementation typically includes a number of reductions and special purpose tests in the beginning that could be thought of in some cases as filter tests. Nevertheless, it is fair to say that the spirit of these two approaches is quite different.

My own preference is the second method. I think it will lead to a cleaner, more understandable, and more maintainable implementation. For this reason, I propose to follow *in the main* the ideas of Pugh in his work on the Omega test at the University of Maryland. In this report, I compare his work with the similar work done about a year earlier by Maydan in his Stanford Thesis. I have, of course, consulted other treatments on dependence analysis, at least enough to convince myself that they would lead to no better results than are presented here. And I also want to express my appreciation to Shin Lee for helpful discussions on this material.

1.3 Terms Used in this Report

Here is an example to illustrate some of the terminology we will be using:

Example 1

do $i = 1,\ n$
  do $j = i,\ m$
    $a(i,\ j) = a(j,\ i{-}1) + 5$
  end do
end do

There are two array references here. Let us call the array reference on the left-hand side array reference 1, and that on the right-hand side array reference 2. (There is no particular significance to how we number the array references; we could have reversed the numbering.) A dependence will exist between these two references if there are integers i_1 and j_1 (the values of i and j in array reference 1) and i_2 and j_2 (the values of i and j in array reference 2) such that we have

Equations:

$$\begin{aligned} i_1 &= j_2 \\ j_1 &= i_2 - 1 \end{aligned}$$

Constraints:

$$\begin{array}{ccccc} 1 & \le & i_1,\, i_2 & \le & n \\ i_1 & \le & j_1 & \le & m \\ i_2 & \le & j_2 & \le & m \end{array}$$

That is, the *equations* come from the subscript expressions and express the fact that the two array references refer to the same memory location. The *constraints*, on the other hand, come from the loop bounds. We refer to the complete set of equations and constraints together as a *dependence problem*.

We always write the constraints using "weak" inequalities (e.g., $\leq$ rather than $<$). In spite of the terminology, this gives tighter results in our subsequent processing. Since all the constants and variables in our equations and constraints are integers, we can turn any strong inequality into a weak one: just replace $a < b$, for instance, by $a + 1 \leq b$.

We distinguish between *exact* and *inexact* tests. An exact test is one that is guaranteed to give an answer of yes or no. (That is, the test reports that either there is or is not a dependence.) Since dependence problems can be arbitrarily complicated, there is no test which is exact for all problems. However, there are tests that are exact for restricted classes of problems. Having an exact test for a dependence problem is the ideal situation that we would like to have.

When an exact test is not available for a dependence problem, we may resort to an inexact test. Such a test reports one of two possible outcomes:

- There is no dependence.
- There may be a dependence.

Of course, in case an inexact test reports that there may be a dependence, the compiler must assume that there *is* a dependence, to be safe. That is, an inexact test gives a conservative approximation to the dependence information.

A typical way that an inexact test may arise is this: We may be able to show that there is a rational solution to the dependence problem, but we may not know if there is an *integral* solution. In this case, the test would report that there may be a dependence.

1.4 Distance and Direction Vectors

Given two array references, which we denote as above by 1 and 2, and associated iteration variable references i_1 and i_2, we define $\Delta i = i_2 - i_1$. We will use this definition consistently in this report.

A direction vector from one array reference to another has the direction vector component $<$ in the position corresponding to the loop with variable i iff the value of i at the source of the dependence vector is less than the value of i at the target. Similarly, the component of the direction distance vector is

$$\text{value of } i \text{ at the source} - \text{value of } i \text{ at the target}$$

Thus:

- if the dependence is from reference 1 to reference 2, the dependence distance is $-\Delta i$, while
- if the dependence is from reference 2 to reference 1, the dependence distance is Δi.

A dependence is meaningful if and only if its distance vector is either

- lexicographically positive, or

- the zero vector, in which case the dependence has to be lexically in the forward direction.

Burke and Cytron refer to such dependences as *plausible* dependences. A plausible dependence is further characterized as

- a **true** or **flow** dependence iff it is a def-use dependence; i.e., if the source is an assignment to the array reference, and the target is a fetch of the value of the array reference.
- a **anti** dependence iff it is a use-def dependence.
- an **output** dependence iff it is a def-def dependence.

It is trivial but useful to note there is a 1-1 correspondence between plausible and non-plausible dependences produced by switching the source and target of the dependence; this transformation thus preserves output dependences and switches true and anti dependences. The transformation has the effect of changing the sign of each component of the distance vector.

1.5 Overview of Processing—Maydan's Approach

Maydan in his thesis proposes the following series of tests for dependence, to be carried out in exactly this order:

SEPARATE Check for separability and reduce to smaller problems if possible. This step uses both the equations and the constraints. (Actually, Maydan does not mention this step, but it is an obvious one.)

GCD Use the "generalized gcd test", partly as an initial filter test, but principally as a way of reducing the equations to a minimal parametrized form. As a test, it is inexact, although it does catch some cases in which there is no dependence. In the rest of the cases it performs the extremely important parametrization step.

This step uses only the equations. After this step, the equations are no longer used, as they have been eliminated by the parametrization.

SINGLE Use the "single variable per constraint test" as an exact test where applicable, and in any case to compute tight bounds on the parameters.

ACYCLIC The acyclic test is an exact test for a small class of problems. It can be used here if appropriate.

RESIDUE The simple loop residue test is an exact test for a small class of problems. It can be used here if appropriate.

FOURIER Finally, if none of the previous tests have given a definitive answer, apply Fourier-Motzkin elimination. This is not an exact test, but Maydan reports that in practice his series of tests is effectively exact—that is, no spurious dependences are reported.

Thus, after the preliminary GCD reduction, Maydan uses a sequence of simple exact tests *only where appropriate*—that is, each test is applied only where it is guaranteed to give an exact result and no further testing will be necessary. Fourier-Motzkin elimination is used at the end to take care of the problems that remain.

1.6 Overview of Processing—Our Approach, Following Pugh

Pugh then showed that the problems for which the Acyclic and Residue tests are appropriate are handled automatically (and equally efficiently) by Fourier-Motzkin elimination; furthermore, the Single Variable Per Constraint Test is also subsumed by his implementation of Fourier-Motzkin elimination. Thus, our final dependence testing algorithm looks like this:

SEPARATE Check for separability and reduce to smaller problems if possible. This step uses both the equations and the constraints. (Pugh does not mention this step either, by the way.)

This step is not, strictly speaking, necessary. It would not change the results of any further analysis if it were omitted. However, it reduces the problem size in some cases, and may therefore lead to faster dependence processing.

GCD Use the "generalized gcd test", exactly as described above.

FOURIER If dependence has not been ruled out by the GCD reduction, apply Fourier-Motzkin elimination with Pugh's "dark shadow" improvement. This subsumes along the way all the rest of Maydan's tests. It is still not an exact test, however, and so Pugh presents a technique (which we call "penumbral calculations" below), amounting to exhaustive search, that can be used to make this test exact. Pugh himself seems uncertain of the extent to which such exhaustive techniques are really necessary; and as mentioned above, Maydan reports that straightforward Fourier-Motzkin elimination has always been exact in practice. (He means by this that in those cases in which it reported a possible dependence, there actually *was* such a dependence, and it was easy to find.) This is probably at least partly due to the extensive exact testing and/or preprocessing that both Pugh and Maydan perform prior to or in the process of using this test.

We shall now present each of Maydan's tests, and then show how Pugh takes account of each of them in his rather simpler version.

2 SEPARATE

A dependence problem is said to be *separable* iff the set of equations and constraints can be separated or divided into two or more disjoint sets in such a way that no two sets have any variables in common. Example 1 is not separable. On the other hand, here is an example that is:

Example 2

do $i = 1,\ 100$
  do $j = 1,\ 50$
    $a(3{*}i{+}2,\ 2{*}j{-}1) = a(5{*}j,\ i{+}3)$
  end do
end do

In this example we have

Equations:

$$\begin{aligned} 3i_1 + 2 &= 5j_2 \\ 2j_1 - 1 &= i_2 + 3 \end{aligned}$$

Constraints:

$$\begin{array}{rcccr} 1 & \le & i_1, i_2 & \le & 100 \\ 1 & \le & j_1, j_2 & \le & 50 \end{array}$$

and the problem separates into the two disjoint subproblems:

Subproblem A:

Equations:

$$3i_1 + 2 = 5j_2$$

Constraints:

$$\begin{array}{rcccr} 1 & \le & i_1 & \le & 100 \\ 1 & \le & j_2 & \le & 50 \end{array}$$

Subproblem B:

Equations:

$$2j_1 - 1 = i_2 + 3$$

Constraints:

$$\begin{array}{rcccr} 1 & \le & i_2 & \le & 100 \\ 1 & \le & j_1 & \le & 50 \end{array}$$

We have to give an algorithm to perform this analysis. Goff, Kennedy, and Tseng give one, but it is flawed in two respects:

- It seems to ignore the constraints and deals only with the equations.
- It apparently does not differentiate between the variables from the two array references (e.g., it does not differentiate i_1 from i_2). So it would not see that Example 2 is separable.

In any case, the algorithm should be straightforward.

There is a trade-off here—the simplification in subsequent processing gained by separating the problem into smaller problems has to be weighed against the cost of doing the separation itself. Certainly an initial implementation can do without this initial step. (In fact, it probably would actually be good to do without it initially, as this would tend to stress the remaining processing and make the implementation more robust.)

3 GCD

3.1 Parametrization

This step, which deals only with the equations and ignores the constraints, has been called the generalized GCD test, because it reduces to the Euclidean algorithm in simple cases. However, it is much more important than just a test—it reduces the number of variables by expressing them in terms of a smaller number of linearly independent parameters. Here is the idea:

First, we write the equations in the form of a matrix equation. For instance, the equations from Example 1 can be written as follows:

$$\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

In general, let us say that the equations of the dependence problem can be written as the matrix equation

$$\mathbf{A}\vec{v} = \vec{c}$$

where $\vec{v}$ is the vector of loop indices (each index giving rise to two components of $\vec{v}$), and $\vec{c}$ is a constant vector ultimately derived from the constant parts of the array subscripts.

What is the shape of **A**? Most of the time, it will be at least as wide as it is high. This just says that there will be at most as many equations as there are variables (where, as we have seen, each loop index gives rise to two variables). There are, to be sure, cases in which this is not true. They all look pretty strange. For instance, a pair of references of the form

$$a(i,\ 4{*}i,\ 7{*}i) \quad \text{and} \quad a(3{*}i{-}1,\ 4{*}i{+}4,\ 5{*}i{+}9)$$

would lead to the matrix equation

$$\begin{pmatrix} 1 & -3 \\ 4 & -4 \\ 7 & -5 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 9 \end{pmatrix}$$

Such an equation is solvable only if the number of linearly independent equations[1] is no more than the number of variables. In this case, that is what happens: the second equation is half the sum of the first and the third.

So one could imagine a first step, which would be needed only if there are more equations than there are variables, that consists of finding a maximal set S of independent equations[2] and checking to make sure that there are no more of these equations than there are variables. The remaining equations can then be discarded—any solution of the equations in S will automatically be a solution of the remaining equations.

Let us assume for the moment that such a procedure has been performed. (It will turn out that we actually don't need a separate procedure to do this—the processing we outline below handles this as a side-effect.)

In any case, we assume temporarily that **A** is at least as wide as it is high. Let us suppose that **A**

[1] Corresponding to each equation with m variables, let us associate an $(m+1)$-dimensional vector whose first m coordinates are the coefficients of the variables in the equation and whose last coordinate is the constant term in the equation. We say that a set of equations is *linearly dependent* if and only if the corresponding set of vectors is linearly dependent.

[2] In this case, any two of the three equations are independent.

is an $n \times m$ matrix, so it looks like this:

$$\begin{array}{cc} & m \text{ columns} \\ \begin{array}{c} n \\ \text{rows} \end{array} & \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \dots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \end{array}$$

where $m \geq n$.

Next—and this is the key idea—we will show that either the equations have no solution or we can find two auxiliary matrices $\mathbf{U}$ and $\mathbf{D}$ such that

- $\mathbf{U}$ is an $m \times m$ unimodular matrix, and in particular is invertible.
- $\mathbf{D}$ is an $n \times m$ lower triangular matrix. Thus, it looks like this:

$$\begin{array}{cc} & m \text{ columns} \\ \begin{array}{c} n \\ \text{rows} \end{array} & \begin{pmatrix} d_{11} & 0 & \dots & 0 & 0 & \dots & 0 \\ d_{21} & d_{22} & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ d_{n1} & d_{n2} & \dots & d_{nn} & 0 & \dots & 0 \end{pmatrix} \end{array}$$

  It has the same shape as $\mathbf{A}$. Further, none of its diagonal elements are 0.
- $\mathbf{AU} = \mathbf{D}$.

We will present an algorithm below for constructing $\mathbf{U}$ and $\mathbf{D}$. The algorithm fails if and only if there is no solution to the equations. In this case, we refer to the processing of the algorithm as the "GCD test". If the algorithm succeeds, the equations have at least one solution, and we refer to the processing of the algorithm as the "GCD parametrization", or the "GCD reduction", for reasons that will become clear below.

Assuming then that we have performed this algorithm successfully and produced $\mathbf{U}$ and $\mathbf{D}$ as above, we have $\mathbf{A} = \mathbf{DU}^{-1}$. Therefore $\mathbf{A}\vec{v} = \vec{c} \iff \mathbf{DU}^{-1}\vec{v} = \vec{c}$, and this holds precisely when there is a vector $\vec{t}$ such that $\vec{v} = \mathbf{U}\vec{t}$ and $\mathbf{D}\vec{t} = \vec{c}$.

So the set of solutions $\vec{v}$ can be found if we can first find the set of all vectors $\vec{t}$ such that $\mathbf{D}\vec{t} = \vec{c}$; the solutions $\vec{v}$ will then just be the vectors $\mathbf{U}\vec{t}$, where $\vec{t}$ runs over this set. Since $\mathbf{U}$ is unimodular, each such vector $\vec{t}$ corresponds uniquely to a solution $\vec{v}$.

Now solving the equation $\mathbf{D}\vec{t} = \vec{c}$ is easy because $\mathbf{D}$ is lower-triangular; the process is conventionally called "forward substitution". When we do this, the first n components of $\vec{t}$ will be uniquely determined, but the last $m - n$ will be arbitrary—this is clear by looking at the form of the matrix $\mathbf{D}$. Thus, $\vec{t}$ runs over an $(m - n)$-dimensional affine subspace of $\mathbf{R}^m$, and so the vectors $\vec{t}$ (and $\vec{v}$ in turn) are parametrized by $m - n$ independent parameters.

So now how do we find the matrices $\mathbf{D}$ and $\mathbf{U}$? The process is in essence that of Gaussian elimination, except that we cannot divide because we want to stay in the domain of integer matrices. We will perform a series of operations on $\mathbf{A}$ that will transform it into $\mathbf{D}$. The operations that we will use are operations on the columns of the matrix (they are called *elementary column operations*), and have the following forms:

interchange Interchange columns i and j.

shear Add an integral multiple of column i to column j.

These operations can be represented as operations on the right by $m \times m$ unimodular matrices:

interchange Interchanging columns i and j amounts to multiplying on the right by the matrix

$$\begin{array}{rc}
 & \begin{array}{ccccccccc} & & \begin{array}{c}\text{column}\\ i\\ \downarrow\end{array} & & & & & \begin{array}{c}\text{column}\\ j\\ \downarrow\end{array} & \end{array} \\
\begin{array}{r} \\ \\ \text{row } i \rightarrow \\ \\ \\ \\ \\ \text{row } j \rightarrow \\ \\ \end{array} & \begin{pmatrix}
1 & & & & & & & & \\
 & 1 & & & & & & & \\
 & & 0 & & & & & 1 & \\
 & & & 1 & & & & & \\
 & & & & 1 & & & & \\
 & & & & & 1 & & & \\
 & & & & & & 1 & & \\
 & & 1 & & & & & 0 & \\
 & & & & & & & & 1
\end{pmatrix}
\end{array}$$

shear Adding b times column i to column j amounts to multiplying on the right by the matrix

$$\begin{array}{rc}
 & \begin{array}{ccccccccc} & & & & & & & \begin{array}{c}\text{column}\\ j\\ \downarrow\end{array} & \end{array} \\
\begin{array}{r} \\ \\ \text{row } i \rightarrow \\ \\ \\ \\ \\ \\ \\ \end{array} & \begin{pmatrix}
1 & & & & & & & & \\
 & 1 & & & & & & & \\
 & & 1 & & & & & b & \\
 & & & 1 & & & & & \\
 & & & & 1 & & & & \\
 & & & & & 1 & & & \\
 & & & & & & 1 & & \\
 & & & & & & & 1 & \\
 & & & & & & & & 1
\end{pmatrix}
\end{array}$$

Here then is what we do:

1. If the first row of $\mathbf{A}$ is all 0, then either the first entry of $\vec{c}$ is also 0 (in which case that row of $\mathbf{A}$ and that entry of $\vec{c}$ can be discarded), or the equation is not solvable; i.e., there is no dependence.

2. So now we assume that the first row of $\mathbf{A}$ is not all 0. Find the entry in the top row of smallest absolute value. Call it α. Interchange columns if necessary so that α is in the leftmost position (i.e., the $(1,1)$ position).

3. Add (or subtract) integer multiples of the first column from the other columns so that the entries at the top of those other columns are each less in absolute value than $|\alpha|$.

There is a fine point here. There are in general *two* such possible values. For instance, if $\alpha = 3$ and the entry at the top of a column is 5, we could wind up with 2 or -1. We might always choose the remainder with smallest absolute value, reasoning that this process is likely to converge faster. It does in general, although not by much. And it is possible that the result it leads to may be either better or worse from the point of further dependence tests. We will see an example of this below. In any case, convergence is so quick, and the problems are generally so trivial, that this doesn't seem to be a significant consideration.

4. Repeat steps 2 and 3 until all entries in the first row except that in position $(1, 1)$ are 0.

That concludes stage 1 of the algorithm. What we have in effect done is used the Euclidean algorithm to find the greatest common divisor of the elements of the first row.

In stage 2, we consider the matrix obtained conceptually by deleting the first row and the first column of our matrix, and repeat this process.

Thus, at the beginning of stage k, the first $k-1$ rows of the matrix have already been put in lower-triangular form, and we are working on the lower-right submatrix composed of those elements whose row and column numbers in the original matrix are each $\geq k$. This is illustrated in Figure 1.

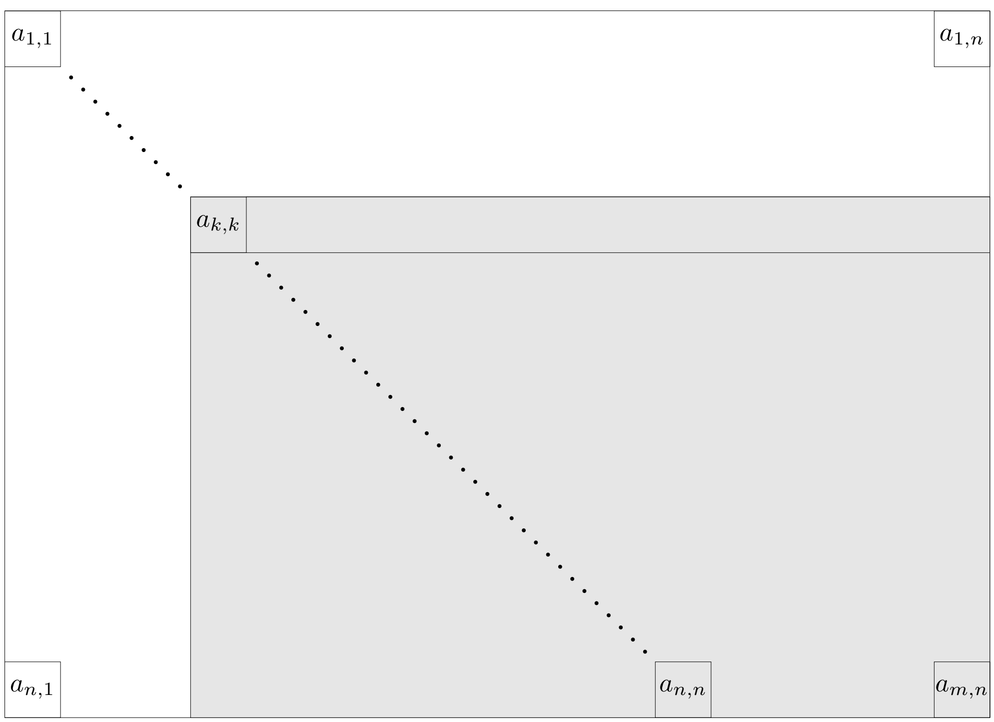

Figure 1: Stage k of the GCD algorithm. The gray area is the submatrix being transformed. The row of elements to the right of $a_{k,k}$ will all become 0 by the end of this stage.

At the beginning of each stage of this process, if the top row of our (sub-) matrix is all 0, then either that row in the full matrix (together with the corresponding element of $\vec{c}$) is a linear combination of previous rows (in which case we delete it), or the original equation is not solvable, i.e., there is no dependence.

Thus, when we have finished this process, we have reduced $\mathbf{A}$ to a lower triangular matrix (as illustrated above) all of whose diagonal elements are non-zero.

Notice that if there were more equations than there were variables (so $\mathbf{A}$ was higher than it was wide; i.e., $n > m$), then in the process outlined above, either we would have thrown away enough redundant equations (i.e., made n smaller) so that $n \leq m$, or we would have discovered an inconsistency in the equations so that we would know that no dependence is possible. Thus, we do not actually have to start with a matrix in which $n \leq m$.

In performing this reduction of $\mathbf{A}$ to $\mathbf{D}$, we can keep track of the operations we have performed on $\mathbf{A}$ by starting with the $m \times m$ identity matrix and performing the identical operations on it. That is, we start with the pair $\langle \mathbf{A}, \mathbf{I}_m \rangle$ and perform identical column operations successively on $\mathbf{A}$ and $\mathbf{I}_m$ until we arrive at two matrices $\langle \mathbf{D}, \mathbf{U} \rangle$. $\mathbf{U}$ is unimodular (since it is the product of unimodular matrices), and by construction, $\mathbf{AU} = \mathbf{D}$. Thus, we have found $\mathbf{U}$ and $\mathbf{D}$ as specified.

The matrices $\mathbf{D}$ and $\mathbf{U}$ are not necessarily uniquely determined. (It is obvious, for one thing, that we have made some arbitrary choices in the algorithm, and it is not obvious that these choices don't lead to different results. In fact, they may.)

However, we *can* apply some more elementary column operations to modify $\mathbf{D}$ if necessary so that

1. all the elements of $\mathbf{D}$ are ≥ 0, and
2. the largest element in each row is the diagonal element.

This is done as follows:

for $i = 1$ **to** n
  if $d_{ii} < 0$ **then**
    Multiply column i by -1. (This is also an elementary column operation.)
    for $j = 1$ **to** $i - 1$
      Add a multiple of column i to column j so that $0 \leq d_{ij} < d_{ii}$
    end for
  end if
end for

After this processing has been performed, $\mathbf{D}$ is said to be in *Hermite normal form.* It can be shown that the Hermite normal form of a matrix is unique.

For our purposes, it is not necessary to put $\mathbf{D}$ in Hermite normal form, since it will not change any of the subsequent computations.

Even if $\mathbf{D}$ is in Hermite normal form, however, $\mathbf{U}$ still does not have to be unique. That is, there may be two unimodular matrices $\mathbf{U_1}$ and $\mathbf{U_2}$ such that

$$\begin{aligned} \mathbf{AU_1} &= \mathbf{D} \\ \mathbf{AU_2} &= \mathbf{D} \end{aligned}$$

Of course this can't happen if $\mathbf{A}$ is non-singular. However, in general $\mathbf{A}$ is singular, since it has more columns than rows. In such a case $\mathbf{U}$ cannot be unique. We can see this as follows: Say we have $\mathbf{AU} = \mathbf{D}$. Since $\mathbf{A}$ is singular, $\mathbf{D}$ must be also. Since $\mathbf{D}$ is in Hermite normal form and is singular, it must have more columns than rows, and hence it has some columns that are all 0. Say $\mathbf{D} = (\mathbf{N_n}|\mathbf{Z})$ where $\mathbf{N_n}$ is an $n \times n$ non-singular diagonal matrix and $\mathbf{Z}$ is an $n \times (m-n)$ zero matrix. That is, $\mathbf{D}$ looks like this:

$$\mathbf{D} = \begin{pmatrix} d_{11} & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & d_{nn} & 0 & \dots & 0 \end{pmatrix}$$

Let $\mathbf{P}$ be a matrix which is the identity on the first n coordinates and is a non-identity unimodular matrix on the last $m-n$ coordinates. (For instance, it could be a non-trivial permutation of the last $m-n$ coordinates.) So $\mathbf{P}$ looks like this:

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & p_{1,1} & \dots & p_{1,m-n} \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & p_{m-n,1} & \dots & p_{m-n,m-n} \end{pmatrix}$$

Then

$$\mathbf{AUP} = \mathbf{DP} = \mathbf{D}$$

so $\mathbf{UP}$ is a unimodular matrix satisfying the same conditions as $\mathbf{U}$ but not equal to it; that is, $\mathbf{U}$ is not unique.

In Sections 4.3 and 4.4 below, we will see an example of the non-uniqueness of $\mathbf{U}$.

3.2 Normalization

After $\mathbf{U}$ and $\mathbf{D}$ have been found, the original constraints can be rewritten in terms of the new variables $\vec{t}$. Each constraint can be written as an inequality of the form

$$a \le f\left(\vec{t}\right) \le b$$

where f is a linear function (i.e., it contains no constant term) with integral coefficients, and a and b are also integers. (It may be that only one of the inequalities is present.)

Now suppose that g is the greatest common divisor of the coefficients of f, chosen so that $g > 0$. Then if f' denotes the function whose coefficients are found by dividing the corresponding coefficients of f by g, we certainly have

$$a/g \le f'\left(\vec{t}\right) \le b/g$$

Now it may be that a/g or b/g are not integers. Since in any case $f'\left(\vec{t}\right)$ is an integer, we must have

$$\lceil a/g \rceil \le f'\left(\vec{t}\right) \le \lfloor b/g \rfloor$$

This process is called *normalization*, and we apply it to each constraint.

In practice, the gcd of the coefficients of f will be 1, and so normalization will really do nothing. Nevertheless, it is worth looking for since it is so cheap to perform. An example of normalization occurs in the analysis of Example 5 (page 21) below.

3.3 Example: GCD as a Parametrization Tool

We'll give two examples of how the GCD test is used. The first example is the triangular loop nest of Example 1. As we have already noted, the equations for this example can be written as the matrix equation

$$\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

That is, $\mathbf{A}\vec{v} = \vec{c}$, with $\vec{v} = \langle i_1, i_2, j_1, j_2 \rangle$, $\vec{c} = \langle 0, -1 \rangle$, and $\mathbf{A}$ being the matrix on the left. Now starting with $\mathbf{A}$ and $\mathbf{I}_4$, we perform successive elementary column operations as follows:

$$\begin{array}{cc} \mathbf{A} & \mathbf{I}_4 \\ \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Add column 1 to column 4:

$$\begin{array}{cc} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Add column 2 to column 3:

$$\begin{array}{cc} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ \mathbf{D} & \mathbf{U} \end{array}$$

Now we solve $\mathbf{D}\vec{t} = \vec{c}$. This gives

$$\vec{t} = \langle 0, 1, t_3, t_4 \rangle$$

Then we have $\vec{v} = \mathbf{U}\vec{t}$. That is,

$$\begin{aligned} i_1 &= t_4 \\ i_2 &= t_3 + 1 \\ j_1 &= t_3 \\ j_2 &= t_4 \end{aligned}$$

and so the possible set of $\vec{v}$ solutions has been represented in terms of the two independent parameters t_3 and t_4. Any values of t_3 and t_4 yield values of $\vec{v}$ that satisfy the equations of this dependence problem, and all such values of $\vec{v}$ are representable in this form.

3.4 Example: GCD as a Dependence Test

This example is from Zima and Chapman, page 168:

Example 3

do $i = 1,\ 100$
$a(i,\ i{+}1) = \ldots$
$\cdots = \ldots a(i{-}1,\ i{-}2) \ldots$
end do

Equations:

$$\begin{aligned} i_1 &= i_2 - 1 \\ i_1 + 1 &= i_2 - 2 \end{aligned}$$

Constraints:

$$1 \quad \leq \quad i_1, i_2 \quad \leq \quad 100$$

With the same conventions as before, we have

$$\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} -1 \\ -3 \end{pmatrix}$$

As before, we compute

$$\begin{array}{cc} \mathbf{A} & \mathbf{I}_2 \\ \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

Now at this point we see that the only way we could have a consistent solution is if the vectors $\langle 1, 0, -1 \rangle$ and $\langle 1, 0, -3 \rangle$ were linearly dependent, which they clearly are not. Hence there is no dependence.

In this case, the GCD test was actually used as a test. This is not too surprising: some dependence problems are really quite simple and are caught immediately by the GCD test.

While the GCD test may seem like a large, computationally expensive algorithm, it really is not: The data structures needed are simple, the typical amount of data to be analyzed is small, and the computation is really nothing more than the Euclidean algorithm, which is quite efficient. (In fact, it's even better than this might indicate: in actual dependence problems, the absolute values of the entries in the matrix are mainly ≤ 1, only sometimes 2, and very rarely greater than 2.)

4 SINGLE—Single Variable Per Constraint Test

Once we have finished with the GCD test, we have eliminated any further need to look at the equations of the dependence problem: all the information in them has been encoded in the parametrization in terms of the $\vec{t}$ components. Everything from this point on deals only with the constraints.

The Single Variable per Constraint Test, like the GCD test, serves two possible functions:

- It may be used as an exact test in certain cases.
- In those cases in which it may not be used as an exact test, it still may be used to give tighter bounds in the constraints. This makes later tests more likely to succeed.

The idea is quite simple. If each constraint involves only one variable, then it defines a upper or lower bound for that variable. Each variable (t_j, say) must be bounded above by the minimum of these upper bounds (call it U_j), and similarly must be bounded below by the maximum of these lower bounds (call it L_j). If for each j, $L_j \leq U_j$, then each t_j can be assigned a value that satisfies all the constraints. Conversely, if there is such an assignment for each t_j, then it must be true that $L_j \leq t_j \leq U_j$.

Thus, in the case in which there is only one variable per constraint, we have only to compute each L_j and U_j and check to see if $L_j \leq U_j$. This test is exact.

4.1 Example: SINGLE as an Exact Test; Finds Dependence

Consider Example 2 on page 7. As we saw, this problem separates into two sub-problems which we have called A and B. The equation for Subproblem A is just

$$\begin{pmatrix} 3 & -5 \end{pmatrix} \begin{pmatrix} i_1 \\ j_2 \end{pmatrix} = -2$$

The GCD algorithm looks like this:

$$\begin{array}{cc}
\mathbf{A} & \mathbf{I}_2 \\
\begin{pmatrix} 3 & -5 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\begin{pmatrix} 3 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \\
\begin{pmatrix} 1 & 3 \end{pmatrix} & \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \\
\begin{pmatrix} 1 & 0 \end{pmatrix} & \begin{pmatrix} 2 & -5 \\ 1 & -3 \end{pmatrix} \\
\mathbf{D} & \mathbf{U}
\end{array}$$

Solving $\mathbf{D}\vec{t} = -2$, we get $\vec{t} = \langle -2, t_2 \rangle$, so $\langle i_1, j_2 \rangle = \mathbf{U}\vec{t}$; i.e.,

$$\begin{aligned} i_1 &= -4 - 5t_2 \\ j_2 &= -2 - 3t_2 \end{aligned}$$

Expressing the constraints in terms of the variable t_2, they become

$$\begin{array}{ccccc} 1 & \leq & -4 - 5t_2 & \leq & 100 \\ 1 & \leq & -2 - 3t_2 & \leq & 50 \end{array}$$

That is,

$$\begin{array}{rcccr} -20 & \leq & t_2 & \leq & -1 \\ -17 & \leq & t_2 & \leq & -1 \end{array}$$

So any value of t_2 in the range $-17 \leq t_2 \leq -1$ will serve to define values of i_1 and j_2 that satisfy the equations and the constraints.

Now we do the same thing for Subproblem B. The equations are

$$\begin{pmatrix} -1 & 2 \end{pmatrix} \begin{pmatrix} i_2 \\ j_1 \end{pmatrix} = 4$$

The GCD algorithm looks like this:

$$\begin{array}{cc} \mathbf{A} & \mathbf{I}_2 \\ \begin{pmatrix} -1 & 2 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} -1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \\ \mathbf{D} & \mathbf{U} \end{array}$$

Solving $\mathbf{D}\vec{t} = 4$ yields $\vec{t} = \langle -4, t_2 \rangle$ so $\langle i_2, j_1 \rangle = \mathbf{U}\vec{t}$; i.e.,

$$\begin{array}{rcl} i_2 & = & -4 + 2t_2 \\ j_1 & = & t_2 \end{array}$$

and we have the constraints

$$\begin{array}{rcccr} 1 & \leq & -4 + 2t_2 & \leq & 100 \\ 1 & \leq & t_2 & \leq & 50 \end{array}$$

Putting the bounds from these two constraints together, we get the final bounds $3 \leq t_2 \leq 50$, so any t_2 in this range will work to define i_2 and j_1 as needed to satisfy the problem. Thus both Subproblems A and B have solutions and so a dependence exists in the original problem.

4.2 Example: SINGLE as an Exact Test; Finds No Dependence

Here is another example, from Goff, Kennedy, and Tseng:

Example 4

```
do i = 1, N
    a(i+2*N) = a(i+N) + C
end do
```

Equations:

$$i_1 + 2N = i_2 + N$$

Constraints:

$$1 \leq i_1,\ i_2 \leq N$$

With the same conventions as before, we have

$$\begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = -N$$

The GCD algorithm looks like this:

$$\begin{array}{cc}
\mathbf{A} & \mathbf{I}_2 \\
\begin{pmatrix} 1 & -1 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\begin{pmatrix} 1 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\
\mathbf{D} & \mathbf{U}
\end{array}$$

Solving $\mathbf{D}\vec{t} = -N$ yields $\vec{t} = \langle -N, t_2 \rangle$. Then $\langle i_1, i_2 \rangle = \mathbf{U}\vec{t}$, i.e.,

$$\begin{array}{rcl} i_1 & = & -N + t_2 \\ i_2 & = & t_2 \end{array}$$

and the constraints are

$$\begin{array}{rcccl} 1 & \le & -N + t_2 & \le & N \\ 1 & \le & t_2 & \le & N \end{array}$$

Equivalently,

$$\begin{array}{rcccr} N+1 & \le & t_2 & \le & 2N \\ 1 & \le & t_2 & \le & N \end{array}$$

This leads immediately to the contradiction

$$N + 1 \le t_2 \le N$$

Therefore, there can be no solution satisfying these constraints, and so there is no dependence.

4.3 SINGLE as a Method of Tightening the Constraints

Even in a situation in which not all the constraints involve a single variable, SINGLE can still be useful. The idea is this: Divide the constraints into two groups:

SV (Single Variable Constraints) These are the constraints involving only a single variable.

MV (Multiple Variable Constraints) These are the constraints involving more than one variable.

First we consider the SV constraints. As before, we use these constraints to get upper and lower bounds for those variables. Following this step, the SV constraints can be discarded, as they are equivalent to the upper and lower bounds thus obtained. The MV constraints, however, will not be discarded or modified in the remainder of the processing.

At this stage, we have two pieces of information:

- The upper and lower bounds.
- The MV constraints.

Next, we look to see if we can substitute the upper and lower bounds just found in the MV constraints. Suppose for instance a particular MV constraint is of the form

$$\sum_i a_i t_i \leq u$$

and t_j is one of the t_i. Then we have

$$a_j t_j \leq u - \sum_{i \neq j} a_i t_i$$

Suppose that for each $i \neq j$ there is a bound (from the SV constraints) of the form

$$\begin{array}{ll} t_i \leq b_i & \text{if } a_i < 0 \\ t_i \geq b_i & \text{if } a_i > 0 \end{array}$$

Then we have

$$a_j t_j \leq u - \sum_{i \neq j} a_i t_i \leq u - \sum_{i \neq j} a_i b_i$$

which yields a further explicit bound for t_j. This bound may be new, or it may be a tightening of a previous bound from the SV constraints. In either case, we merge it with those bounds, and we iterate this process until no further tightening of the bounds is obtained. Note again that the MV constraints are never changed: at each stage in this process, successively tighter bounds are substituted in the original MV constraints. One of two things will happen:

- Either a contradiction will emerge, or
- Tighter bounds will be found, which can be profitably used in later tests. (It may be, of course, that the original bounds derived from the SV constraints cannot be tightened further by this method. In this case, this process concludes in one step.)

For example, suppose we have the constraints

$$\begin{array}{lcccc} (1) & 0 & \leq & 2t_1 + t_2 & \\ (2) & & & 3t_1 + 4t_2 & \leq & 12 \end{array}$$

and suppose we know to begin with that

$$-5 \leq t_1$$

Substituting this bound in inequality (2) yields

$$t_2 \leq 6$$

Continuing in this manner, we get

$$\begin{array}{rcccc} -3 & \leq & t_1 & & \\ & & t_2 & \leq & 5 \\ -2 & \leq & t_1 & & \\ & & t_2 & \leq & 4 \end{array}$$

and after this point, no further progress can be made. Thus, we have arrived at the bounds

$$\begin{array}{rcl} -2 & \leq & t_1 \\ & & t_2 \leq 4 \end{array}$$

which we can store for later use. Figure 2 shows graphically how this successive approach to better and better bounds works.

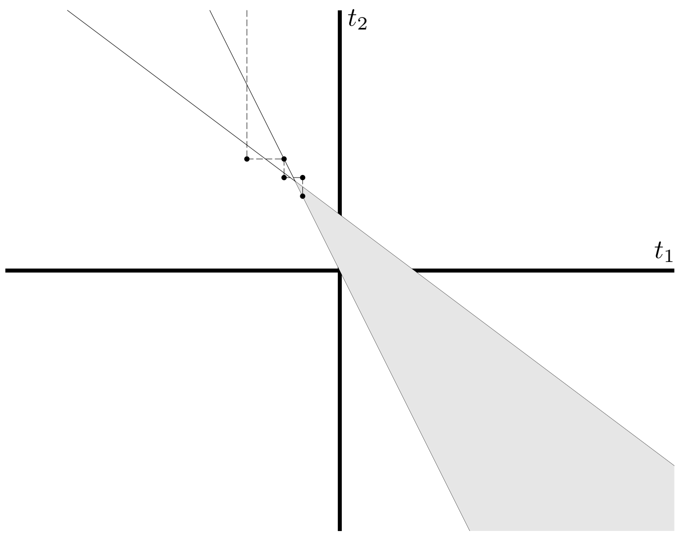

Figure 2: SINGLE as a method of refining upper and lower bounds.

Here is a rather contrived example from Banerjee, page 135. It is instructive in that it provides a good workout for the GCD reduction and the Single Variable Per Constraint Test.

Example 5

do $i = 0,\ 20$
  do $j = 0,\ 20$
    $a(3{*}i{-}2{*}j{-}1,\ 4{*}i{-}2{*}j{-}4) = \dots$
    $\dots = \dots a(2{*}i{+}2{*}j{+}1,\ -3{*}i{-}6{*}j{+}3)\dots$
  end do
end do

Equations:

$$\begin{array}{rcl} 3i_1 - 2j_1 - 1 & = & 2i_2 + 2j_2 + 1 \\ 4i_1 - 2j_1 - 4 & = & -3i_2 - 6j_2 + 3 \end{array}$$

Constraints:

$$\begin{array}{rcccl} 0 & \leq & i_1, i_2 & \leq & 20 \\ 0 & \leq & j_1, j_2 & \leq & 20 \end{array}$$

The problem is not separable. The equations become

$$\begin{pmatrix} 3 & -2 & -2 & -2 \\ 4 & 3 & -2 & 6 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \end{pmatrix}$$

Figure 3 shows the GCD processing for this problem.

Solving $\mathbf{D}\vec{t} = \vec{c}$ amounts to solving the equations

$$\begin{aligned} t_1 &= 2 \\ 7t_1 + t_2 &= 7 \end{aligned}$$

So we have

$$\vec{t} = \langle 2, -7, t_3, t_4 \rangle$$

and then $\vec{v} = \mathbf{U}\vec{t}$ is the general solution. That is,

$$\begin{aligned} i_1 &= 2 + 2t_4 \\ i_2 &= 23 + 8t_3 + 6t_4 \\ j_1 &= -7 - 3t_3 + t_4 \\ j_2 &= -14 - 5t_3 - 4t_4 \end{aligned}$$

The constraints then become

$$\begin{array}{lcccccc} \textbf{(SV1)} & \quad & 0 & \leq & 2 + 2t_4 & \leq & 20 \\ \textbf{(MV1)} & & 0 & \leq & 23 + 8t_3 + 6t_4 & \leq & 20 \\ \textbf{(MV2)} & & 0 & \leq & -7 - 3t_3 + t_4 & \leq & 20 \\ \textbf{(MV3)} & & 0 & \leq & -14 - 5t_3 - 4t_4 & \leq & 20 \end{array}$$

and after normalization these are

$$\begin{array}{lcrcccr} \textbf{(SV1)} & \quad & -1 & \leq & t_4 & \leq & 9 \\ \textbf{(MV1)} & & -11 & \leq & 4t_3 + 3t_4 & \leq & -1 \\ \textbf{(MV2)} & & 7 & \leq & -3t_3 + t_4 & \leq & 27 \\ \textbf{(MV3)} & & 14 & \leq & -5t_3 - 4t_4 & \leq & 34 \end{array}$$

(Notice the tightening of the bounds in MV1 due to the application of the floor and ceiling in the normalization.)

Constraint SV1 is the only SV constraint. It provides upper and lower bounds for t_4. Substituting these bounds in the remaining MV constraints yields

$$\begin{array}{rcccr} -38 & \leq & 4t_3 & \leq & 2 \\ -2 & \leq & -3t_3 & \leq & 28 \\ 10 & \leq & -5t_3 & \leq & 70 \end{array}$$

$$\mathbf{A} \qquad \mathbf{I}_4$$

$$\begin{pmatrix}3 & -2 & -2 & -2\\ 4 & 3 & -2 & 6\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Interchange columns 1 and 2:

$$\begin{pmatrix}-2 & 3 & -2 & -2\\ 3 & 4 & -2 & 6\end{pmatrix} \qquad \begin{pmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Add column 1 to column 2; subtract it from columns 3 and 4:

$$\begin{pmatrix}-2 & 1 & 0 & 0\\ 3 & 7 & -5 & 3\end{pmatrix} \qquad \begin{pmatrix}0 & 1 & 0 & 0\\ 1 & 1 & -1 & -1\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Interchange columns 1 and 2:

$$\begin{pmatrix}1 & -2 & 0 & 0\\ 7 & 3 & -5 & 3\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 0\\ 1 & 1 & -1 & -1\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Add 2 times column 1 to column 2:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 7 & 17 & -5 & 3\end{pmatrix} \qquad \begin{pmatrix}1 & 2 & 0 & 0\\ 1 & 3 & -1 & -1\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Interchange columns 2 and 4:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 7 & 3 & -5 & 17\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 2\\ 1 & -1 & -1 & 3\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\end{pmatrix}$$

Add 2 times column 2 to column 3; subtract 6 times column 2 from column 4:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 7 & 3 & 1 & -1\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 2\\ 1 & -1 & -3 & 9\\ 0 & 0 & 1 & 0\\ 0 & 1 & 2 & -6\end{pmatrix}$$

Interchange columns 2 and 3:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 7 & 1 & 3 & -1\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 2\\ 1 & -3 & -1 & 9\\ 0 & 1 & 0 & 0\\ 0 & 2 & 1 & -6\end{pmatrix}$$

Subtract 3 times column 2 from column 3; add column 2 to column 4:

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 7 & 1 & 0 & 0\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0 & 2\\ 1 & -3 & 8 & 6\\ 0 & 1 & -3 & 1\\ 0 & 2 & -5 & -4\end{pmatrix}$$

$$\mathbf{D} \qquad \mathbf{U}$$

Figure 3: GCD processing for Example 5

That is,

$$\begin{array}{rcccr} -9 & \leq & t_3 & \leq & 0 \\ -9 & \leq & t_3 & \leq & 0 \\ -14 & \leq & t_3 & \leq & -2 \end{array}$$

and so we have

$$-9 \leq t_3 \leq -2$$

Now we go back and substitute these bounds into constraints MV1, MV2, and MV3. After simplification, we get

$$-1 \leq t_4 \leq 7$$

We continue this process, getting successively

$$\begin{array}{rcccr} -8 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 6 \\ -7 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 5 \\ -6 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 4 \\ -5 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 2 \\ -4 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 1 \\ -3 & \leq & t_3 & \leq & -2 \\ -1 & \leq & t_4 & \leq & 0 \\ -2 & \leq & t_3 & \leq & -3 \end{array}$$

which is impossible. Therefore there is no dependence in this problem. This derivation can be seen graphically in Figure 4. In that figure, two of the three constraint pairs are almost parallel; this is part of what causes the procedure to take so long.

4.4 Same Problem, Modified Computations

When performing the GCD reductions in the previous problem, we adopted the technique of using the remainder of smallest absolute value at each step. Now let us perform the same algorithm, but use the smallest *positive* remainder at each step. The computation is shown in Figure 5.

The number of steps here was the same as in the original calculation—nothing was really lost by not taking the remainder with the least absolute value.

Now we continue: As before, the solution to $\mathbf{D}\vec{t} = \vec{c}$ is

$$\vec{t} = \langle 2, -7, t_3, t_4 \rangle$$

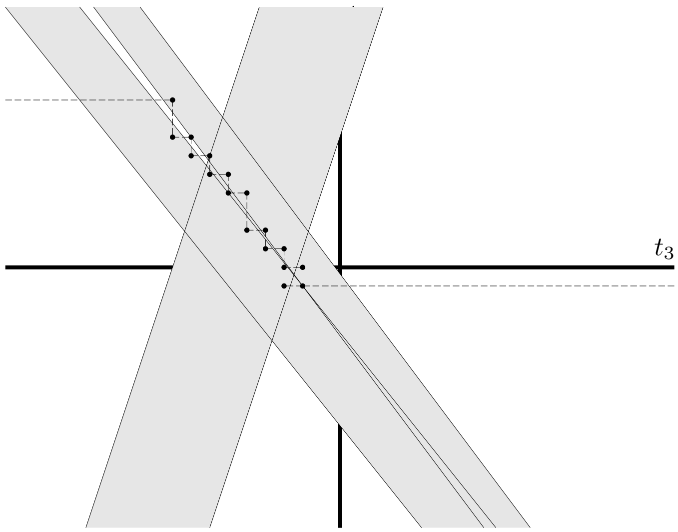

Figure 4: SINGLE computations arriving at a contradiction. The upper and lower bounds for t_3 cross after 15 iterations.

and the general solution is $\vec{v} = \mathbf{U}\vec{t}$. That is,

$$\begin{aligned} i_1 &= 2 + 2t_4 \\ i_2 &= 23 + 8t_3 + 14t_4 \\ j_1 &= -7 - 3t_3 - 2t_4 \\ j_2 &= -63 - 5t_3 - 5t_4 \end{aligned}$$

The constraints then become

$$\begin{array}{lrclcl} \textbf{(SV1)} & 0 & \leq & 2 + 2t_4 & \leq & 20 \\ \textbf{(MV1)} & 0 & \leq & 23 + 8t_3 + 14t_4 & \leq & 20 \\ \textbf{(MV2)} & 0 & \leq & -7 - 3t_3 - 2t_4 & \leq & 20 \\ \textbf{(MV3)} & 0 & \leq & -63 - 5t_3 - 5t_4 & \leq & 20 \end{array}$$

Again, constraint SV1 is the only SV constraint. As before, it becomes

$$-1 \leq t_4 \leq 9$$

Now substituting these bounds in the MV constraints yields

$$\begin{array}{rcccr} -18 & \leq & t_3 & \leq & 1 \\ -15 & \leq & t_3 & \leq & -2 \\ -25 & \leq & t_3 & \leq & -12 \end{array}$$

so we have

$$\mathbf{A} \qquad \mathbf{I}_4$$

$$\begin{pmatrix} 3 & -2 & -2 & -2 \\ 4 & 3 & -2 & 6 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 1 and 2:

$$\begin{pmatrix} -2 & 3 & -2 & -2 \\ 3 & 4 & -2 & 6 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 1 to column 2; subtract it from columns 3 and 4:

$$\begin{pmatrix} -2 & 1 & 0 & 0 \\ 3 & 7 & -5 & 3 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 1 and 2:

$$\begin{pmatrix} 1 & -2 & 0 & 0 \\ 7 & 3 & -5 & 3 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Add 2 times column 1 to column 2:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 7 & 17 & -5 & 3 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & 0 & 0 \\ 1 & 3 & -1 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 2 and 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 7 & 3 & -5 & 17 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 2 \\ 1 & -1 & -1 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \qquad \text{Same computation up to here.}$$

Add 2 times column 2 to column 3; subtract 5 times column 2 from column 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 7 & 3 & 1 & 2 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 2 \\ 1 & -1 & -3 & 8 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 2 & -5 \end{pmatrix} \qquad \text{This is the step that differs.}$$

Interchange columns 2 and 3:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 7 & 1 & 3 & 2 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 2 \\ 1 & -3 & -1 & 8 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & -5 \end{pmatrix}$$

Subtract 3 times column 2 from column 3 and 2 times column 2 from column 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 7 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 2 \\ 1 & -3 & 8 & 14 \\ 0 & 1 & -3 & -2 \\ 0 & 2 & -5 & -9 \end{pmatrix}$$

$$\mathbf{D} \qquad \mathbf{U}$$

Figure 5: Modified GCD processing for Example 5

$$-15 \leq t_3 \leq -12$$

Another substitution gives us

$$6 \leq t_4 \leq 8$$

Yet another substitution gives us

$$-14 \leq t_3 \leq -19$$

so as before, we get a contradiction. This time, however, we only had to substitute 3 times, instead of 15. Figure 6 shows graphically what happens. Here the three constraint pairs form three regions that are quite distinct, and we can see how the process arrives quickly at a contradiction.

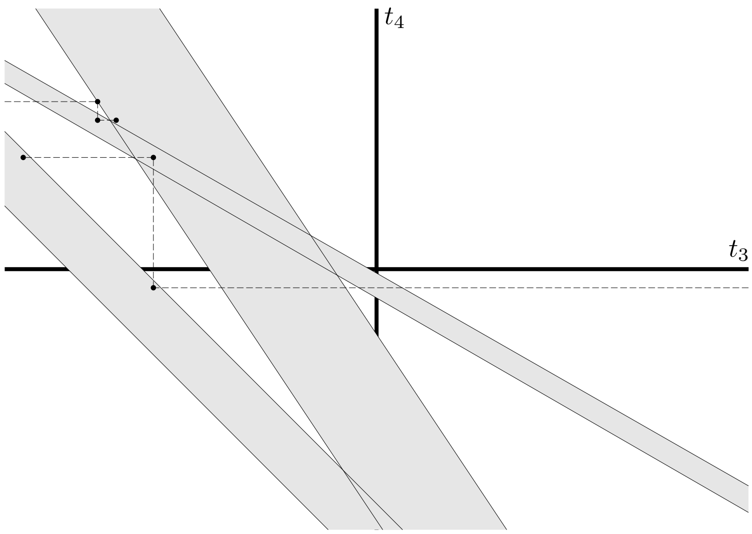

Figure 6: SINGLE computations arriving at a contradiction. The upper and lower bounds for t_3 cross after 3 iterations.

Thus, the "optimized" modulus used in the first GCD reduction did not actually lead to a quicker solution for **U** and **D**, and it actually led to worse performance in the following Single-Variable Constraint Test. Of course, this was a pretty strange test to begin with; I don't expect such behavior to occur in practice.

Again, the point is simply that optimizing the modulus operation in the GCD reduction doesn't seem to be worth worrying about.

5 ACYCLIC—The Acyclic Test

If we have got to this point, then the set MV of constraints containing more than one variable is not empty. In addition, each variable may or may not have upper or lower bounds derived from the single-variable constraints. Let us call this set of bounds B. As we have noted, solutions of the MV constraints that also satisfy the bounds B correspond exactly to the solutions of the original problem.

We may now try to apply the Acyclic Test. This test applies if and only if there is a variable that is only constrained in one direction *by the constraints in MV*. What this means is this: for any variable (t_i, say), we can write all the constraints in MV that involve t_i in the form

$$\begin{aligned} a_{1i}t_i &\leq f_1\left(\vec{t}\right) \\ a_{2i}t_i &\leq f_2\left(\vec{t}\right) \\ &\cdots \\ a_{ri}t_i &\leq f_r\left(\vec{t}\right) \end{aligned}$$

where each $f_j\left(\vec{t}\right)$ is an affine function not involving the component t_i. If when we do this all the coefficients a_{ji} have the same sign, then we say that t_i is constrained in only one direction[3].

So say t_i is constrained in only one direction. Let us first assume that all $a_{ji} > 0$. Let us also assume that there is a lower bound lb_i for t_i in B.

Now rewrite the constraints above, substituting lb_i for t_i. Certainly if there is a solution to these new constraints that satisfies the rest of the constraints in MV and the bounds in B, then this solution satisfies the original set of constraints. And conversely, if there is any solution to the original set of constraints, then since it satisfies the inequalities above, it also must satisfy those same inequalities with t_i replaced by lb_i.

If t_i has no lower bound in B, then the inequalities above can be simply discarded from the set MV—a solution to the problem without these constraints can always be extended to a solution to the complete problem by making t_i sufficiently small (i.e., sufficiently negative).

If all the a_{ji} are < 0, the same procedure can be used, except that now we use the upper bound of t_i from B, if it exists.

Thus in any case we have replaced our problem by a new one with at least some constraints containing fewer variables, and having a solution if and only if the original problem has one.

Now it may be that this reduced problem has some single-variable constraints in it. We can thus iterate the Single Variable per Constraint Test with the Acyclic Test. We do this until

- SINGLE shows that solution either does or does not exist, or
- No further improvement can be made, and there remains at least one constraint in MV.

Maydan gives the following example of the use of the Acyclic Test:

$$\begin{array}{lrcccr} \textbf{(SV1)} & 1 & \leq & t_1 & \leq & 10 \\ \textbf{(SV2)} & 1 & \leq & t_2 & \leq & 10 \\ \textbf{(SV3)} & 0 & \leq & t_3 & \leq & 4 \end{array}$$

$$\begin{array}{lrcl} \textbf{(MV1)} & t_2 & \leq & t_1 \\ \textbf{(MV2)} & t_1 & \leq & t_3 + 4 \end{array}$$

t_2 is only constrained above by the MV constraints, and it has a lower bound of 1 from the SV constraints. Therefore we can replace MV1 by $1 \leq t_1$. This is now a single-variable constraint; it is actually subsumed by SV1, although in general it might be added to or modify the SV constraints.

[3] Again, note that this pertains only to the MV constraints. There may well be—and usually are—SV constraints that constrain t_i in the other direction

(Note that the SV constraints are really what we have been denoting by B, since they are solved for their respective variables.) Thus we have a reduced problem

$$
\begin{array}{lrcccl}
\textbf{(SV1)} & 1 & \le & t_1 & \le & 10 \\
\textbf{(SV2)} & 1 & \le & t_2 & \le & 10 \\
\textbf{(SV3)} & 0 & \le & t_3 & \le & 4 \\
\textbf{(MV2)} & & & t_1 & \le & t_3 + 4
\end{array}
$$

Now in this problem, t_1 is only constrained above by the MV constraints, and it has a lower bound of 1 from the SV constraints. Therefore, we can replace MV2 by $1 \le t_3 + 4$. That is $-3 \le t_3$, which is subsumed by SV3. Thus, we have reduced the problem again, this time to

$$
\begin{array}{lrcccl}
\textbf{(SV1)} & 1 & \le & t_1 & \le & 10 \\
\textbf{(SV2)} & 1 & \le & t_2 & \le & 10 \\
\textbf{(SV3)} & 0 & \le & t_3 & \le & 4
\end{array}
$$

and since this problem plainly has solutions, so does the original one.

Maydan claims this test is actually useful, but does not give an actual example in his thesis. In any case, we will show below (following Pugh) how it is efficiently subsumed by Fourier-Motzkin elimination.

6 RESIDUE—The Simple Loop Residue Test

If we have reached this point and we are not done, there must be at least some constraints that are cyclic. The Simple Loop Residue Test, developed by Pratt, applies to cyclic constraints that are all of the form

$$t_i \le t_j + c \tag{1}$$

The idea is that if we have a sequence of such constraints starting and ending at t_1, say:

$$
\begin{array}{ccc}
t_1 & \le & t_2 + c_1 \\
t_2 & \le & t_3 + c_2 \\
& \cdots & \\
t_n & \le & t_1 + c_n
\end{array}
$$

then these constraints collapse into the resulting constraint

$$t_1 \le t_1 + \sum_{i=1}^{n} c_i$$

which can be true if and only if $\sum c_i \ge 0$. In fact, if *all* the remaining constraints are of the form (1), then they can all be satisfied simultaneously if and only if all the sums from the corresponding cycles are ≥ 0. (This is because the variables t_j could be assigned any values whatsoever if this is true.)

Consider Example 1 (page 4). We saw on page 15 that the possible set of solutions can be parametrized as

$$\begin{aligned} i_1 &= t_4 \\ i_2 &= t_3 + 1 \\ j_1 &= t_3 \\ j_2 &= t_4 \end{aligned}$$

The constraints are then expressed in terms of t_3 and t_4, and become

$$\begin{array}{lcccccc}
\textbf{(SV1)} & 1 & \le & t_4 & & \\
\textbf{(SV2)} & 1 & \le & t_3+1 & & \\
\textbf{(SV3)} & & & t_4 & \le & m \\
\textbf{(SV4)} & & & t_3+1 & \le & m \\
\textbf{(MV1)} & t_4 & \le & t_3 & & \\
\textbf{(MV2)} & t_3+1 & \le & t_4 & & \\
\textbf{(SV5)} & & & t_3 & \le & n \\
\textbf{(SV6)} & & & t_4 & \le & n
\end{array}$$

The Single Variable per Constraint Test might apply, if we have a pretty smart dependence analyzer: the SV constraints can be used successively to show that the upper bounds for t_3 and t_4, which start out at $\min\{m-1, n\}$ and $\min\{m, n\}$ respectively, can be reduced to $\min\{m-j-1, n\}$ and $\min\{m-j, n\}$ at the j^{th} step. Since j is arbitrary, we can conclude that the upper bound for each variable is $-\infty$. This is pretty sophisticated reasoning for a poor little computer program, however.

The Acyclic Test does not apply, because t_3 and t_4 form a cycle.

The Simple Loop Residue Test, however, applies at once: we have

$$t_4 \le t_3 \le t_4 - 1$$

which is impossible. Therefore, no dependence can exist. This test really is a short-circuited version of the sophisticated reasoning indicated above using the Single Variable per Constraint Test. Of course the Single Variable per Constraint Test applies in many cases for which the Simple Loop Residue Test cannot be used; it's just that in this special case, the Simple Loop Residue Test is both quicker and much less sophisticated in its application.

Maydan points out that if a cyclic constraint is of the form

$$at_i \le at_j + c$$

then a can be divided out (using a floor or ceiling for c/a as usual), and so such constraints can also be handled by this test.

6.1 Example: A Skewed Nearest-Neighbor Problem

This is a standard nearest-neighbor type of problem, as would be found in solving a discretized partial differential equation. We use a skewed form as considered in Goff, Kennedy, and Tseng:

Example 6

```
do i = 1, n
   do j = i+1, n+1
      a(i, j−i) = a(i−1, j−1) + a(i, j−1+1) + a(i+1, j−1) + a(i, j−1+1)
   end do
end do
```

We'll just consider the first term on the right-hand side:

Equations:

$$\begin{aligned} i_1 &= i_2 - 1 \\ j_1 - i_1 &= j_2 - i_2 \end{aligned}$$

Constraints:

$$\begin{array}{rcccl} 1 & \leq & i_1 & \leq & n \\ 1 & \leq & i_2 & \leq & n \\ 1 + i_1 & \leq & j_1 & \leq & n + i_1 \\ 1 + i_2 & \leq & j_2 & \leq & n + i_2 \end{array}$$

This problem is not separable. The equations are represented as

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

The GCD reduction proceeds as follows:

$$\begin{array}{cc}
\mathbf{A} & \mathbf{I}_4 \\[6pt]
\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 1 & -1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\[6pt]
\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 \end{pmatrix} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\[6pt]
\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\[6pt]
\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\[6pt]
\mathbf{D} & \mathbf{U}
\end{array}$$

Solving $\mathbf{D}\vec{t} = \vec{c}$, we get $\vec{t} = \langle -1, -1, t_3, t_4 \rangle$. $\vec{i} = \mathbf{D}\vec{t}$ then yields

$$\begin{aligned} i_1 &= -1 + t_3 \\ i_2 &= t_3 \\ j_1 &= -1 + t_4 \\ j_2 &= t_4 \end{aligned}$$

The constraints then become

$$\begin{array}{rcccl} 1 & \leq & -1 + t_3 & \leq & n \\ 1 & \leq & t_3 & \leq & n \\ t_3 & \leq & -1 + t_4 & \leq & n - 1 + t_3 \\ 1 + t_3 & \leq & t_4 & \leq & n + t_3 \end{array}$$

That is,

$$\begin{array}{lrcccl} \textbf{(SV1)} & 2 & \leq & t_3 & & \\ \textbf{(SV2)} & & & t_3 & \leq & n+1 \\ \textbf{(SV3)} & 1 & \leq & t_3 & & \\ \textbf{(SV4)} & & & t_3 & \leq & n \\ \textbf{(MV1)} & & t_3 & \leq & -1 + t_4 & \\ \textbf{(MV2)} & & t_4 & \leq & n - 1 + t_3 & \\ \textbf{(MV3)} & & t_3 & \leq & -1 + t_4 & \\ \textbf{(MV4)} & & t_4 & \leq & n + t_3 & \end{array}$$

Some of these inequalities are subsumed by others; we reduce the set of inequalities to this:

$$\begin{array}{lrcccl} \textbf{(SV1)} & 2 & \leq & t_3 & & \\ \textbf{(SV4)} & & & t_3 & \leq & n \\ \textbf{(MV1)} & & t_3 & \leq & -1 + t_4 & \\ \textbf{(MV2)} & & t_4 & \leq & n - 1 + t_3 & \end{array}$$

We can substitute the SV bounds in the MV constraints, yielding

$$\begin{array}{rcccl} & & t_4 & \leq & 2n - 1 \\ 3 & \leq & t_4 & & \end{array}$$

which is consistent, provided that $n \geq 2$. However, this does not constitute an exact test. In this case, the Simple Loop Residue Test is exact: the two MV constraints form a loop, and we get

$$t_3 \leq t_4 - 1 \leq n - 2 + t_3$$

which shows that there is a dependence, provided again that $n \geq 2$.

7 FOURIER—Fourier-Motzkin Elimination

At this point, we apply Fourier-Motzkin elimination to the remaining constraints. This is a technique, going back to Fourier (and subsequently rediscovered by Dines and later by Motzkin), for successively eliminating variables from the constraints. It works like this: Say the variables are

$\{t_1, t_2, \ldots, t_n\}$. Pick one of the variables represented in an MV constraint—say we pick t_1—and solve all the constraints for that variable, dividing out by its coefficient. (Note that we cannot throw away the remainders at this point—this is *not* integer division. We have to keep the quotients as rational numbers.) We get three sets of inequalities, which we will call the A inequalities:

$$\begin{array}{ccc} t_1 \geq D_1(\bar{t}) & t_1 \leq E_1(\bar{t}) & 0 \leq F_1(\bar{t}) \\ \ldots & \ldots & \ldots \\ t_1 \geq D_p(\bar{t}) & t_1 \leq E_q(\bar{t}) & 0 \leq F_r(\bar{t}) \end{array}$$

where now $\bar{t} = (t_2, \ldots, t_n)$ and D_i, E_i, and F_i are affine functions of $\bar{t}$ with rational coefficients.

Note that there will be both inequalities involving functions D_i and E_j: if there were no D_i inequalities, then t_1 would only be constrained above by the MV constraints, and so would already have been eliminated by the processing of the Acyclic Test; and similar reasoning would apply if there were no E_j inequalities.

Suppose that these inequalities have a solution $(t_1, \bar{t})$. Then certainly

$$\max_i D_i(\bar{t}) \leq t_1 \leq \min_j E_j(\bar{t})$$

Thus, if the inequalities have a solution $(t_1, \bar{t})$, then we must have what we will call the B inequalities:

$$\begin{aligned} \max_i D_i(\bar{t}) &\leq \min_j E_j(\bar{t}) \\ 0 &\leq \min_k F_k(\bar{t}) \end{aligned}$$

Conversely, if the B inequalities have a solution $\bar{t}$, then there is a value t_1 such that

$$\max_i D_i(\bar{t}) \leq t_1 \leq \min_j E_j(\bar{t})$$

and so the A inequalities are satisfied. In other words, we have shown that there is a solution $(t_1, \bar{t})$ to the A inequalities if and only if there is a solution $\bar{t}$ to the B inequalities. In this way, we have reduced the problem by one dimension, by eliminating the variable t_1; and we can continue in this fashion until either a contradiction is reached or we have proved that a solution actually exists.

In practice, we don't do things quite in this way:

- We don't actually compute the max and min as above. (These would be symbolic expressions in any case.) Instead, we replace the original "D" and "E" inequalities with all possible equalities of the form
$$D_i(\bar{t}) \leq E_j(\bar{t})$$
- We don't reduce the coefficients of t_1 to 1, since that would involve representing rational numbers in our implementation. Instead, we normalize the coefficients of t_1 to be the least common multiple of the original coefficients. That way, all the coefficients in the inequalities remain integers.

There are a couple of things to note at this point:

1. Fourier-Motzkin elimination is an exact method for finding rational solutions of inequalities. As it stands, however, it can show one of two possibilities:

   - There are no rational solutions, and therefore there are no integer solutions.
   - There is a rational solution, and therefore there might be an integer solution.

   Thus Fourier-Motzkin elimination is in general an inexact dependence test. There are many cases, however, in which the test can be shown to be exact. We will see examples of this below.

2. The elimination step, although it reduces the number of variables by 1, can cause the number of inequalities to increase. The increase can be explosive: If there are n equations in d variables, in the worst case one can wind up with about

   $$\left(\frac{n}{4}\right)^{2^d}$$

   equations by the time all the variables have been eliminated. This has caused the algorithm to be ignored until recently. It does appear, however, that there are efficient implementations that make this test practical for dependence problems that occur in practice.

Geometrically, Fourier-Motzkin elimination amounts to projection onto the hyperplane that is perpendicular to the variable being eliminated. For example, suppose we have the two inequalities

$$4x + 5y \leq 20$$
$$3x + 5y \geq 15$$

Solving for x, we get the inequalities

$$12x \leq -15y + 60$$
$$12x \geq -20y + 60$$

and so by eliminating x, we get

$$-20y + 60 \leq -15y + 60$$

or

$$0 \leq y$$

Figure 7 illustrates this problem. The reduced inequality $0 \leq y$ is represented by the thickened shadow, and is the projection of the original region onto the y-axis.

Note that in this case, the shadow includes the integer point $(0, 1)$, but the original region does not include a integer point that projects onto this point. Because of this phenomenon, the shadow produced by Fourier-Motzkin elimination is referred to as the "real shadow"; the term "real" is used here to indicate that the shadow consists of solutions over the real numbers (in fact, over the rational numbers), as opposed to the integers.

Since we really need integer solutions for dependence analysis, Fourier-Motzkin elimination does not in general constitute an exact test. It can be made exact in every case, however, by a rather expensive process of exhaustive search where needed. We will describe this process below in the section on the "penumbra".

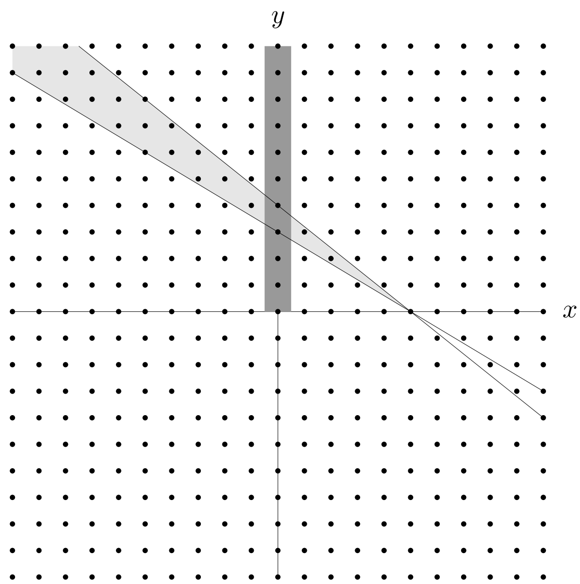

Figure 7: Fourier-Motzkin elimination applied to x. The projection on the y-axis is shown as a (thickened) shadow.

7.1 Example: A Triangular Loop Nest

This example is from Banerjee (page 133):

Example 7

do $i = 1,\ 100$
  do $j = i,\ 50$
    $a(i{-}2{*}j{-}60) = \ldots$
    $\cdots = \ldots a(-i{-}j{+}50) \ldots$
  end do
end do

Equation:

$$i_1 - 2j_1 - 60 = -i_2 - j_2 + 50$$

Constraints:

$$\begin{array}{rcccl} 1 & \leq & i_1,\, i_2 & \leq & 100 \\ i_1 & \leq & j_1 & \leq & 50 \\ i_2 & \leq & j_2 & \leq & 50 \end{array}$$

This is a 1-dimensional problem, so there is no question of separability. The equations are just

$$\begin{pmatrix}1 & 1 & -2 & 1\end{pmatrix}\begin{pmatrix}i_1\\ i_2\\ j_1\\ j_2\end{pmatrix} = 110$$

The usual GCD reduction then gives

$$\underset{\mathbf{D}}{\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}} \qquad \underset{\mathbf{U}}{\begin{pmatrix}1 & -1 & 2 & -1\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}}$$

Solving $\mathbf{D}\vec{t} = c$ gives $\vec{t} = \langle 110, t_2, t_3, t_4\rangle$. The solution is then $\vec{v} = \mathbf{U}\vec{t}$. That is,

$$\begin{aligned} i_1 &= 110 - t_2 + 2t_3 - t_4 \\ i_2 &= t_2 \\ j_1 &= t_3 \\ j_2 &= t_4 \end{aligned}$$

The constraints then become

$$\begin{array}{rcccl} 1 & \le & 110 - t_2 + 2t_3 - t_4 & \le & 100 \\ 1 & \le & t_2 & \le & 100 \\ 110 - t_2 + 2t_3 - t_4 & \le & t_3 & \le & 50 \\ t_2 & \le & t_4 & \le & 50 \end{array}$$

and we can write these as follows:

$$\begin{array}{lrcccl} \textbf{(SV1)} & 1 & \le & t_2 & \le & 100 \\ \textbf{(SV2)} & & & t_3 & \le & 50 \\ \textbf{(SV3)} & & & t_4 & \le & 50 \\ \textbf{(MV1)} & 1 & \le & 110 - t_2 + 2t_3 - t_4 & \le & 100 \\ \textbf{(MV2)} & 110 - t_2 + 2t_3 - t_4 & \le & t_3 & & \\ \textbf{(MV3)} & t_2 & \le & t_4 & & \end{array}$$

Substituting the SV bounds into the MV constraints yields the tighter set of bounds

$$\begin{array}{lrcccr} \textbf{(SV1)} & 1 & \le & t_2 & \le & 50 \\ \textbf{(SV2)} & -53 & \le & t_3 & \le & -10 \\ \textbf{(SV3)} & 1 & \le & t_4 & \le & 50 \end{array}$$

There are no acyclic constraints here, so we skip the Acyclic Test. Similarly, the Simple Loop Residue Test does not apply. We proceed to Fourier-Motzkin elimination: Solving the MV constraints for t_2, we get

$$t_2 \le 109 + 2t_3 - t_4$$

$$t_2 \le t_4$$

$$t_2 \ge 10 + 2t_3 - t_4$$

$$t_2 \ge 110 + t_3 - t_4$$

When we consider these equations in pairs, we find that the first and fourth yield

$$110 + t_3 \le 109 + 2t_3$$

which amounts to $t_3 \ge 1$, and we know this cannot be true—it violates a previously computed bound. Thus we have proved that there is no dependence. Note that the preliminary tightening of the loop bounds helped us here.

Incidentally, if we had decided to eliminate t_3 instead of t_2, we would have proceeded like this:

$$2t_3 \ge -109 + t_2 + t_4$$

$$2t_3 \le -10 + t_2 + t_4$$

$$2t_3 \le -220 + 2t_2 + 2t_4$$

The first and third equations yield

$$t_2 + t_4 \ge 111$$

and applying the upper bound for t_4 to this inequality, we get $t_2 \ge 61$, which we know violates the upper bound on t_2, so again we see that there is no dependence.

Starting with t_4 would be entirely similar to starting with t_2.

7.2 Example: Fourier-Motzkin Elimination as an Exact Test

This example comes from Wolfe (page 258, problem 7.9):

Example 8

```
do i = 1, 10
   do j = i, 10
      a(i+8*j+3) = a(i+8*j−6)
   end do
end do
```

Equation:

$$i_1 + 8j_1 + 3 = i_2 + 8j_2 - 6$$

Constraints:

$$\begin{array}{rcccl} 1 & \le & i_1, i_2 & \le & 10 \\ i_1 & \le & j_1, j_2 & \le & 10 \end{array}$$

The problem is 1-dimensional; there is no question of separability. The GCD reduction gives

$$\begin{array}{rcl} i_1 & = & -9 + t_2 - 8t_3 + 8t_4 \\ i_2 & = & t_2 \\ j_1 & = & t_3 \\ j_2 & = & t_4 \end{array}$$

and the constraints become

$$\begin{array}{lrccll} \textbf{(SV1)} & 1 & \le & t_2 & \le & 10 \\ \textbf{(SV2)} & 1 & \le & t_3 & \le & 10 \\ \textbf{(SV3)} & 1 & \le & t_4 & \le & 10 \\ \textbf{(MV1)} & 1 & \le & -9 + t_2 - 8t_3 + 8t_4 & \le & 10 \end{array}$$

No further tightening of the bounds is possible at this point. The Acyclic and Simple Loop Residue tests do not apply, so we proceed to Fourier-Motzkin elimination: we have

$$t_2 \le 19 + 8t_3 - 8t_4$$
$$t_2 \le 10$$
$$t_2 \ge 10 + 8t_3 - 8t_4$$
$$t_2 \ge 1$$

Combining these equations in pairs yields a couple of inequalities that are trivially always true, and the following inequality:

$$-18 \le 8t_3 - 8t_4 \le 0$$

Now we have previously mentioned that this elimination step might in principle lose information. That is, there might be a solution of this reduced set of inequalities that does not correspond to a solution with an integral value of t_2. However, in this case, this is impossible. The reason is that the coefficients of t_2 in the above inequalities were all 1. Reverting back to our original formulation on page 33, the values of D_i and E_j are therefore integers. Hence if

$$\max_i D_i(\bar{t}) \le \min_j E_j(\bar{t})$$

there must be an integer between them. In other words, *a Fourier-Motzkin elimination step is exact when the coefficients of the variable being eliminated are all 1.* Note that this was *not* the case in the example in Figure 7.

Now let us proceed. Dividing out by the greatest common factor (4) of the coefficients of t_3 and t_4 in the inequality just produced, we have reduced our problem to the following set of inequalities:

$$\begin{array}{ccccc} -2 & \le & t_3 - t_4 & \le 0 \\ 1 & \le & t_3 & \le 10 \\ 1 & \le & t_4 & \le 10 \end{array}$$

The only MV constraint here is the first, and the Acyclic Test shows at once that it is consistent. Therefore, a dependence exists.

Thus, at each step of the Fourier-Motzkin elimination, it may pay to revisit previous tests in our sequence of tests to see if they apply.

Alternatively, we could simply proceed with a second Fourier-Motzkin elimination step on t_3, say. Since all the coefficients of t_3 are 1, this step will be exact. When we do this, we get

$$1 \le t_4 \le 12$$

and this is consistent with the final MV constraint on t_4. Equivalently, to carry the Fourier-Motzkin elimination to the end, we eliminate t_4 (which again has coefficient 1) from the inequality

$$1 \le t_4 \le 10$$

and get $1 \le 10$, which is certainly true.

7.3 The Dark Shadow

Let us again consider the problem of inexactness in Fourier-Motzkin elimination. We have already seen that if all the coefficients of t_1 are 1, then eliminating t_1 is exact. Suppose that we have two constraints of the form

$$\begin{array}{ccccc} D(\bar{t}) & \leq & t_1 & & \\ & & at_1 & \leq & E(\bar{t}) \end{array}$$

where $a > 1$. Then we have

$$aD(\bar{t}) \leq at_1 \leq E(\bar{t})$$

and the projection would consist of the inequality

$$aD(\bar{t}) \leq E(\bar{t})$$

Now it turns out that this projection too is exact: for there to be an integral solution t_1 to the original inequality, it is necessary and sufficient that there be a multiple of a lying between aD and E. Since aD is itself a multiple of a, we see that an integral t_1 exists if and only if

$$aD(\bar{t}) \leq E(\bar{t})$$

Thus, *a Fourier-Motzkin elimination step is exact when, for each two paired inequalities, the coefficient of the variable being eliminated is 1 in at least one of them.*

So the only case we have to consider is that in which the inequalities are of the form

$$\begin{array}{ccccc} D(\bar{t}) & \leq & at_1 & & \\ & & bt_1 & \leq & E(\bar{t}) \end{array}$$

and a and b are each greater than 1. Figure 7 provides an example of this.

We rewrite the inequality as

$$bD(\bar{t}) \leq abt_1 \leq aE(\bar{t})$$

(We could have rewritten it so that the coefficient of t_1 became the least common multiple of a and b; the conclusion we reach below would not change, and the way we have done it here is a bit simpler.)

Now one way to guarantee a solution is this: suppose there is an integral vector $\bar{t}$ for which $bD(\bar{t}) + ab \leq aE(\bar{t})$. Then the region being projected has a thickness $\geq ab$ at that point and therefore must include an integer of the form abt_1 (i.e., it must include a multiple of ab). Thinking of the region being projected as a translucent object, the shadow at this point (i.e., at $\bar{t}$) would be relatively dark, since the region is relatively thick.

Actually, Pugh has a slightly weaker condition having the same effect. It works like this:

For there to be no multiple of ab between $bD(\bar{t})$ and $aE(\bar{t})$, one would have to have, for some integer i,

$$abi < bD(\bar{t}) \leq aE(\bar{t}) < ab(i+1) \tag{2}$$

This could not be true if

$$aE(\bar{t}) - bD(\bar{t}) \geq ab \tag{3}$$

This is just what we saw above. Pugh, however, points out that since all the terms in the inequality (2) are integers, and since the first two terms are multiples of b while the last two are multiples of a, it is equivalent to the inequality

$$abi + b \leq bD(\bar{t}) \leq aE(\bar{t}) \leq ab(i+1) - a \tag{4}$$

and this inequality cannot be satisfied if

$$aE(\bar{t}) - bD(\bar{t}) \geq ab - a - b + 1 \tag{5}$$

and this is weaker than the inequality (3). So if we replace the inequality $bD(\bar{t}) \leq aE(\bar{t})$ in the Fourier-Motzkin elimination process by the inequality (5), we will get a stronger condition. Pugh refers to the the set of values of $\bar{t}$ satisfying these stronger conditions as the *dark shadow*. The significant facts are these:

- If the dark shadow is not empty, it must contain an integer vector.
- Each integer vector in the dark shadow is guaranteed to come from an integer vector in the original region.
- If either a or b in the above derivation is 1, then the dark shadow is the same as the real shadow. This is evident from inequality (5). This shows that the dark shadow is not too conservative an approximation; we have seen above that this *should* be true.

Thus, we can say that if the real shadow is contains no integer points (in particular, if it is empty), there can be no dependence. If the dark shadow contains an integer point, there is definitely a dependence. There is still the possibility that the real shadow contains integer points but the dark shadow contains none; that is, all the integer points in the real shadow lie outside the dark shadow. These points may or may not correspond to integer points in the original region.

For a simple example, consider the inequalities

$$4x + 5y \leq 20$$
$$3x + 5y \geq 15$$

from Figure 7, which were rewritten as

$$12x \leq -15y + 60$$
$$12x \geq -20y + 60$$

Inequality (5) for the dark shadow becomes

$$(-15y + 60) - (-20y + 60) \geq 12 - 3 - 4 + 1$$

That is, $y \geq 2$. Figure 8 shows the real shadow (as before), together with the dark shadow.

Note that there are two integer points in the real shadow that are not in the dark shadow. One of them—the point (0,0)—comes from an integer point in the original region, while the other—the point (0,1)—does not.

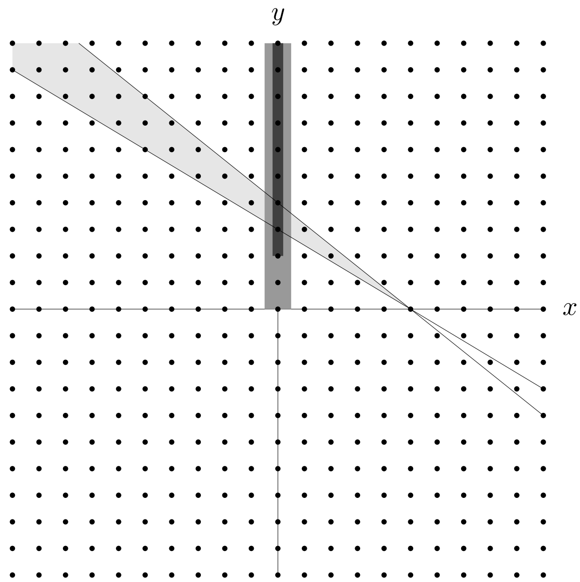

Figure 8: Fourier-Motzkin elimination applied to x. Now we also show the dark shadow.

7.4 The Penumbra

Let us refer to those points in the real shadow but not in the dark shadow as the *penumbra*. Continuing with the notation we have been using, an integer point $\bar{t}$ in the real shadow will correspond to a solution $(t_1, \bar{t})$ coming from the penumbra if and only if

$$\begin{array}{rcccll} 0 & \leq & aE(\bar{t}) - bD(\bar{t}) & \leq & ab - a - b & \quad \bar{t} \text{ is not in the dark shadow} \\ bD(\bar{t}) & \leq & abt_1 & \leq & aE(\bar{t}) & \quad (t_1, \bar{t}) \text{ is an integer solution} \end{array}$$

In particular, then, we must have

$$bD(\bar{t}) \leq abt_1 \leq bD(\bar{t}) + ab - a - b$$

and so

$$D(\bar{t}) \leq at_1 \leq D(\bar{t}) + \left\lfloor \frac{ab - a - b}{b} \right\rfloor$$

Thus, it is enough to search in this region. We do this by setting an integer i successively to the numbers

$$0, 1, \ldots, \left\lfloor \frac{ab - a - b}{b} \right\rfloor$$

and *replacing* the inequality

$$D(\bar{t}) \leq at_1$$

with the equality

$$at_1 = D(\bar{t}) + i$$

in the original constraints. There is an integer solution to the original problem with $\bar{t}$ in the penumbra if and only if there is an integer solution to one of these "penumbral problems"[4].

Notice that if $a > b$ (remember that we have fixed things so that a and b are both positive), we would instead successively substitute the alternative equalities

$$bt_1 = E(\bar{t}) - i$$

where i runs over the smaller interval

$$0, 1, \ldots, \left\lfloor \frac{ab - a - b}{a} \right\rfloor$$

for the original inequality

$$bt_1 \leq E(\bar{t})$$

In any case, we have shown that there is a solution to the original problem if and only if either

- The dark shadow is not empty, or
- There is a solution to one of the penumbral problems.

Let us return to the problem we have been considering, which is illustrated in Figure 7:

$$4x + 5y \leq 20$$
$$3x + 5y \geq 15$$

The first step was to eliminate x. When we did this, we found the values $y = 0$ and $y = 1$ in the penumbra. The coefficients of x are 3 and 4 (a and b, respectively), so we replace the second inequality successively by the equations

$$3x = 15 - 5y + i$$

for $0 \leq i \leq 1$, and we check the values of 0 and 1 for y. Let us see how this works:

$(i = 0)$ $3x = 15 - 5y$. $y = 0$ yields the solution $x = 5$, which is compatible with the first inequality. $y = 1$ is impossible because 3 does not divide $15 - 5y = 10$.

$(i = 1)$ $3x = 16 - 5y$. 3 does not divide $16 - 5y$ for either $y = 0$ or $y = 1$.

So we see that $(x, y) = (5, 0)$ is a solution whose image is in the penumbra, and is the only such solution.

These computations are illustrated in Figure 9.

So in principle, Fourier-Motzkin elimination can be made an exact test as follows:

- Successively eliminate variables in the standard way.

[4] Pugh refers to these penumbral problems as "splinters".

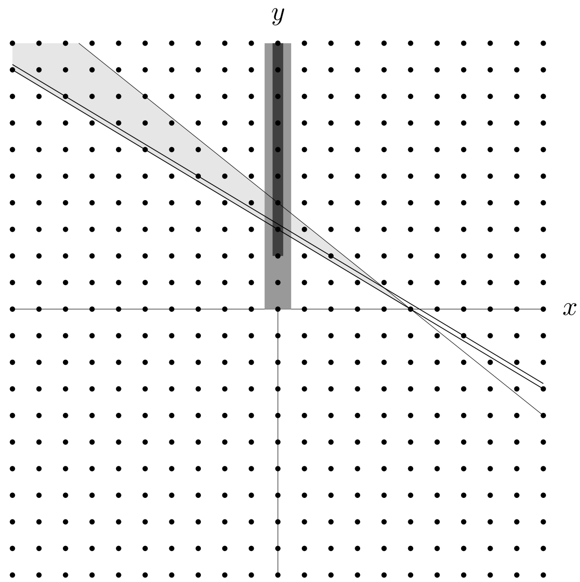

Figure 9: The penumbral computations. The two dark closely parallel lines correspond to the equations

$$3x = 15 - 5y$$

and

$$3x = 16 - 5y$$

- At each step, see if the elimination is exact. This amounts to seeing if the dark shadow is all of the real shadow. This will certainly be true if all the coefficients of the variable being eliminated are ± 1. So if possible, choose such a variable to be eliminated at each step.
- At the final step,

1. If the real shadow contains no integer points (in particular, if it is empty, there is no dependence.
2. Otherwise, if there is at least one integer point in the dark shadow, there is a dependence.
3. If the dark shadow was equal to the real shadow at each step, we are done. Otherwise, go back, and for each step at which the dark shadow was not equal to the real shadow, create a family of additional tests corresponding to the penumbra for that step, and repeat the process with those tests.

In practice, this is almost certainly overkill, and looks to be quite expensive. I would think that a good first implementation would simply treat Fourier-Motzkin elimination as an inexact test—that is, report a possible dependence at step 3 of the final step above whenever the elimination is not exact at any step. This could then be extended by adding the penumbra tests at the last step only; my guess is that this would catch almost all the remaining examples.

8 Which Tests Are Really Necessary?

Pugh has pointed out that a good implementation of Fourier-Motzkin elimination subsumes the processing of the SINGLE, ACYCLIC, and RESIDUE tests. Here we will show how this happens. We will treat the ACYCLIC and RESIDUE tests first, because what is going on there is quite simple.

8.1 Eliminating the Acyclic Test

In the example on page 28, Fourier-Motzkin elimination applied to the variable t_2 would yield (by an exact projection) the constraint

$$1 \le t_1$$

and then eliminating t_1 would yield (again exactly)

$$-3 \le t_3$$

These are precisely the same computations as were performed by the Acyclic Test.

The Acyclic Test would cause us to perform the elimination in precisely this manner: t_2 first, then t_1. On the other hand, if we just performed Fourier-Motzkin elimination on t_1 first, the resulting computations would be no more complex.

Further, each Fourier-Motzkin elimination step corresponding to a use of the Acyclic Test will be exact, because the SV lower or upper bound that is used has in effect a coefficient of 1 for the variable being eliminated.

There is one part of the Acyclic Test that has to be kept in mind, however: When there is no SV bound for a variable that is constrained in only one direction by its MV bounds, those MV bounds are discarded by the Acyclic Test, because they can be automatically satisfied, provided all the other constraints can be satisfied; we mentioned this in the discussion of the Acyclic Test. Pugh does exactly this as part of his processing of the constraints before applying Fourier-Motzkin elimination.

8.2 Eliminating the Residue Test

This test only applies when the coefficients of the variables are all 1. Hence the Fourier-Motzkin projections are all exact, and amount to precisely the same computations as are performed by the Residue Test.

8.3 Eliminating the Single Variable Per Constraint Test

Looking at the processing in this test in terms of Fourer-Motzkin projections enables us to make a significant improvement in the computations. Let us write each constraint in the form "*expression* ≥ 0"; in fact, we can then leave off the "≥ 0", as it is redundant, and represent each constraint simply as an expression. A linear combination formed by adding together positive multiples of such expressions is another usable expression—this just expresses the fact that a sum of non-negative numbers is non-negative. Thus, if in two such expressions a variable occurs with coefficients of opposite sign, that variable can be eliminated from a linear combination of those two expressions.

If in addition the coefficient of the variable in at least one of the two original expressions was ± 1, that elimination will correspond to an exact projection. (This is all simply a matter of representation; it just happens to be a convenient way to manage the inequalities. In itself, it adds nothing to the analysis.)

Let us consider first the example on page 20. The constraints we are given in this example then are represented as the two expressions

$$2t_1 + t_2 \qquad (1)$$
$$-\,3t_1 - 4t_2 + 12 \qquad (2)$$

Now since the coefficient of t_2 in equation (1) is 1, we can eliminate t_2 by an exact Fourier-Motzkin projection to get

$$5t_1 + 12$$

which is normalized to

$$t_1 + 2 \qquad (3)$$

Constraint (3) can in turn be used with (2) (since the coefficients of t_1 have opposite signs) to eliminate t_1 by an exact projection, giving

$$-\,4t_2 + 18$$

which is normalized to

$$-\,t_2 + 4 \qquad (4)$$

Note that *in normalizing an expression, the constant term is always rounded down* (i.e. towards $-\infty$). This is just because if $x \geq 0$, then it must also be true that $\lfloor x \rfloor \geq 0$, and the second inequality is stronger.

In this way, we have arrived at the same bounds ($-2 \leq t_1$; $t_2 \leq 4$) as in our original computation. The projections were both exact in this case (and in fact, the point $(-2, 4)$ is in the original shaded region).

Finally, we did not even need to “seed” our computation with the a priori bound $-5 \leq t_1$, as we did on page 20. We could have, of course; this *a priori* bound would simply have been represented as the expression $t_1 + 5$, but it would not have improved our final result. For instance, combining this expression with (2) would yield the (normalized) expression $-t_2 + 6$. This expression, however, is a weaker bound that that given by (4), and so can be discarded.

This is an example of a general phenomenon: *if two expressions have the same coefficients for corresponding variables, then the one with the lower constant term subsumes the other.* This is because, as in the reasoning above, it represents a tighter bound.

So now we can see that there was really nothing particularly special about using the SV constraints in the SINGLE processing. The point of an SV constraint simply was that the coefficient of the (single) variable was 1, and therefore, substituting that bound in the MV constraints amounted to an exact projection. The same thing would happen, however, with any MV constraint containing a variable whose coefficient was ± 1. Using these constraints can greatly speed up the processing.

Let us consider, for example, the computations on page 25. We can write the constraints as follows:

$-t_4+9$	(1)
t_4+1	(2)
$-4t_3-7t_4-2$	(3)
$4t_3+7t_4+11$	(4)
$3t_3+2t_4+27$	(5)
$-3t_3-2t_4-7$	(6)
t_3+t_4+16	(7)
$-t_3-t_4-13$	(8)

Now the constraints with unitary coefficients for t_4 are (1), (2), (7), and (8). We can pair each of them with the constraints having coefficients for t_4 of opposite signs. In this way, we get the following expressions:

$(1,4)$	$4t_3+74$	$\Longrightarrow$	t_3+18	(9)	subsumed by 10
$(1,5)$	$-3t_3+45$	$\Longrightarrow$	t_3+15	(10)	
$(1,7)$	t_3+25				subsumed by 10
$(2,3)$	$-4t_3+5$	$\Longrightarrow$	$-t_3+1$	(11)	subsumed by 12
$(2,6)$	$-3t_3-5$	$\Longrightarrow$	$-t_3-2$	(12)	subsumed by 13
$(2,7)$	$-t_3-12$			(13)	(subsumed by 14)

(We use parentheses in the expression "(subsumed by 14)" to indicate that this expression is subsumed by an expression to be produced in a later pass. We will follow this convention consistently.) So far, this is the same as what we did originally, since we have just used the SV constraints (1) and (2). Now, however, we can go on and use (7) and (8):

$(7,3)$	$3t_3+110$	$\Longrightarrow$	t_3+36		subsumed by 10
$(7,6)$	$-t_3+25$				subsumed by 13
$(8,4)$	$-3t_3-80$	$\Longrightarrow$	$-t_3-27$	(14)	inconsistent with 10

and so we arrive at a contradiction rather more quickly than we did originally.

For comparison, let us look at the same problem, but with a different GCD reduction (using the version of mod that returns the remainder of least absolute value). This was presented on page 22, and took considerably longer to stabilize. The constraints there now get written like this:

$-t_4+9$	(1)
t_4+1	(2)
$-4t_3-3t_4-1$	(3)
$4t_3+3t_4+11$	(4)
$3t_3-t_4+27$	(5)
$-3t_3+t_4-7$	(6)
$5t_3+4t_4+34$	(7)
$-5t_3-4t_4-14$	(8)

Here the constraints with unitary coefficients for t_4 are (1), (2), (5), and (6). Proceeding as before, we get

$(1,4)$	$4t_3 + 38$	$\Longrightarrow t_3 + 9$	(9)	(subsumed by 12)
$(1,6)$	$-3t_3 + 2$	$\Longrightarrow -t_3$	(10)	subsumed by 11
$(1,7)$	$5t_3 + 70$	$\Longrightarrow t_3 + 14$		subsumed by 9
$(2,3)$	$-4t_3 + 2$	$\Longrightarrow -t_3$		same as 10
$(2,5)$	$3t_3 + 28$	$\Longrightarrow t_3 + 9$		same as 9
$(2,8)$	$-5t_3 - 10$	$\Longrightarrow -t_3 - 2$	(11)	(subsumed by 13)

So far, the processing is the same as that performed originally, since we have just used the SV constraints (1) and (2). Now as before, we go on to use (5) and (6):

$(5,4)$	$13t_3 + 92$	$\Longrightarrow t_3 + 7$	(12)	
$(5,7)$	$17t_3 + 142$	$\Longrightarrow t_3 + 8$		subsumed by 12
$(6,3)$	$-13t_3 - 22$	$\Longrightarrow -t_3 - 2$		same as 11
$(6,8)$	$-17t_3 - 42$	$\Longrightarrow -t_3 - 3$	(13)	

Up to this point, the tightest constraints—12 and 13—are still consistent. At this point, we can go on to perform inexact projections using constraints 3, 4, 7, and 8:

$(3,7)$	$-t_3 + 98$	subsumed by 13
$(4,8)$	$t_3 + 2$	inconsistent with 13

And so again we arrive at a contradiction. That is, we have shown that the real shadow is empty, and so there can be no integer solutions. This process converged *much* faster than the original set of calculations, although it was still slower than the alternate version done immediately previously. So again, using Pugh's modified mod operator cannot be said to lead always to an improvement in performance, and, in this case at least, actually slowed down the computation.

9 Computing Direction and Distance Vectors

Pugh has also shown how Fourier-Motzkin elimination can be used to determine direction and distance vectors: introduce a new variable for each direction distance; for instance, introduce $\Delta i = i_2 - i_1$. Then perform the GCD reduction in such a way that these variables are not eliminated. This is done by placing these variables at the end of the variable vector, and never interchanging columns of $\mathbf{A}$ that correspond to these variables. Each of these variables will then be one of the t_j variables produced by the GCD reduction, and Fourier-Motzkin elimination can then be used to find the corresponding bounds.

Burke and Cytron's hierarchical technique for analyzing multi-dimensional dependences can be used to organize the computations. This works by successively searching down a tree which looks like that in Figure 10 in the case of a three-dimensional iteration space.

We will illustrate this with three examples.

9.1 Example: The Skewed Nearest-Neighbor Problem Again

We reconsider Example 6 (page 31). We write the equations as

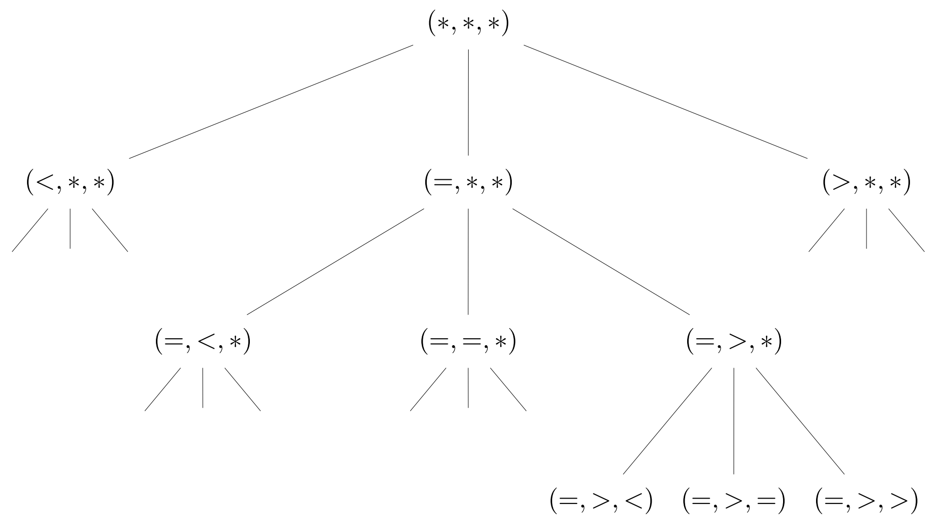

Figure 10: Burke-Cytron Search Tree for Multi-Dimensional Dependence Testing

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \\ \Delta i \\ \Delta j \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The GCD reduction becomes

$$\begin{array}{cc} \mathbf{A} & \mathbf{I}_6 \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Add column 1 to column 2:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 2 and 3:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 2 to column 4:

$$\underset{\mathbf{D}}{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}} \qquad \underset{\mathbf{U}}{\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}$$

Note how we left the last two columns of matrix $\mathbf{D}$ alone. Now solving

$$\mathbf{D}\vec{t} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

yields

$$\vec{t} = \langle -1, -1, t_3, t_4, 1, 1 \rangle$$

and then $\vec{v} = \mathbf{U}\vec{t}$ gives

$$\begin{aligned} i_1 &= t_3 - 1 \\ i_2 &= t_3 \\ j_1 &= t_4 - 1 \\ j_2 &= t_4 \\ \Delta i &= 1 \\ \Delta j &= 1 \end{aligned}$$

Thus if we take the dependence to be a true dependence (i.e., from reference 1 on the left to reference 2 on the right), the direction vector will be $(<, <)$, and in fact the distance vector will be $(-1, -1)$. Proving that this dependence actually exists is straighforward using Fourier-Motzkin elimination (as we have already mentioned), and we omit the details.

9.2 Example: A Triangular Iteration Space

This example is taken from a prepublication draft of a book[5] by Allen and Kennedy, who deal with this using a "triangular Banerjee inequality". Our processing is much simpler.

[5] "Advanced Compilation for Vector and Parallel Computers", to be published by Morgan Kaufman Publishers. The draft is available at `ftp://softlib.rice.edu/pub/Kennedy/book`.

Example 9

do $i = 1,\ 100$
  do $j = 1,\ i{-}1$
    $a(j) = a(i{+}j{-}1) + C$
  end do
end do

Equations:

$$\begin{array}{rcl} j_1 & = & i_2 + j_2 - 1 \\ \Delta i & = & i_2 - i_1 \\ \Delta j & = & j_2 - j_1 \end{array}$$

Constraints:

$$\begin{array}{rcccl} 1 & \leq & i_1,\, i_2 & \leq & 100 \\ 1 & \leq & j_1 & \leq & i_1 - 1 \\ 1 & \leq & j_2 & \leq & i_2 - 1 \end{array}$$

The equations are then written like this:

$$\begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \\ \Delta i \\ \Delta j \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Note that the variables Δi and Δj are placed at the "far end" of the variable vector. This corresponds to the last two columns of the array $\mathbf{A}$. Now we proceed with the usual GCD reduction:

$$\begin{array}{cc} \mathbf{A} & \mathbf{I}_6 \\ \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Interchange the first two columns:

$$\begin{array}{cc} \begin{pmatrix} 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Add column 1 to column 3; subtract it from column 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 2 to column 3; subtract it from columns 4 and 5:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & -1 & -1 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 3 to column 4; subtract it from column 6:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 & -1 & -1 \\ 1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{D} \qquad\qquad\qquad \mathbf{U}$$

Note that at no point did we interchange any column with one of the last two columns—the columns corresponding to Δi and Δj. Now solving $\mathbf{D}\vec{t} = \left(\begin{smallmatrix}1\\0\\0\end{smallmatrix}\right)$ yields

$$\vec{t} = \langle 1, 1, 0, t_4, t_5, t_6 \rangle$$

and then $\vec{v} = \mathbf{U}\vec{t}$. That is,

$$\begin{aligned} i_1 &= 1 - t_5 - t_6 \\ i_2 &= 1 - t_6 \\ j_1 &= t_4 - t_6 \\ j_2 &= t_4 \\ \Delta i &= t_5 \\ \Delta j &= t_6 \end{aligned}$$

The constraints are now written as the following expressions:

$-t_5 - t_6$	(1)	(subsumed by 9)
$99 + t_5 + t_6$	(2)	
$-t_6$	(3)	(subsumed by 10)
$99 + t_6$	(4)	
$t_4 - t_6 - 1$	(5)	
$-t_5 - t_4$	(6)	
$t_4 - 1$	(7)	
$-t_6 - t_4$	(8)	

Since the variables we really care about are t_5 ($=\Delta i$) and t_6 ($=\Delta j$), we first eliminate t_4:

$$\begin{array}{rllll}
(5,6) & -t_5-t_6-1 & & (9) & \\
(5,8) & -2t_6-1 & \Longrightarrow -t_6-1 & (10) & \\
(7,6) & -t_5-1 & & (11) & \\
(7,8) & -t_6-1 & & & \text{same as 10}
\end{array}$$

So at this point we see from expressions 10 and 11 that t_5 and t_6 are both ≤ -1. Thus the only possible dependence vector is $(<,<)$. Continuing, we can eliminate t_5 (the expressions remaining are 2, 4, 9, 10, and 11):

$$\begin{array}{rlll}
(2,9) & 98 & & \\
(2,11) & 98+t_6 & & (12)
\end{array}$$

The expressions remaining now are 10 and 12 (4 is subsumed by 12). This yields $-98 \leq t_6 \leq -1$. Since all the eliminations were exact, there really are dependences. Each dependence is an anti dependence (from reference 2 to reference 1) with dependence vector $(<,<)$.

9.3 Example: A Four-Dimensional Iteration Space

Our next example is one mentioned by Pugh, which Wonnacott says is one of the most complicated that they encountered in a large suite of standard benchmark tests.

Example 10

do $j = 0,\ 20$
  do $i = \max(-j,\ -10),\ 0$
    do $k = \max(-j,\ -10) - i,\ -1$
      do $l = 0,\ 5$
        $a(l,\ i,\ j) = \ldots a(l,\ k,\ i{+}j) \ldots$
      end do
    end do
  end do
end do

In writing the equations and constraints for this dependence problem, we are going to switch[6] the convention that we have been using: the variables from the left-hand side array reference will have subscript 2 and those from the right-hand side array reference will have subscript 1.

9.3.1 Separation: The First Problem

The first thing to notice about this dependence problem is that it is separable–the equations and constraints for l are independent of those for the other variables. So the first dependence problem consists simply of the

Equations:

$$\begin{array}{rcl}
l_2 & = & l_1 \\
\Delta l & = & l_2 - l_1
\end{array}$$

[6]It's just because I worked it out this way and didn't want to redo all the calculations.

Constraints:

$$\begin{array}{rcll} 0 & \le & l_1 & \le 5 \\ 0 & \le & l_2 & \le 5 \end{array}$$

which is clearly solvable, and we have $l_2 = l_1$; i.e., $\Delta l = 0$.

9.3.2 The GCD Reduction

The second problem is composed of the rest of the variables, and is the significant one: we have Equations:

$$\begin{array}{rcl} i_2 & = & k_1 \\ j_2 & = & i_1 + j_1 \\ \Delta i & = & i_2 - i_1 \\ \Delta j & = & j_2 - j_1 \\ \Delta k & = & k_2 - k_1 \end{array}$$

Constraints:

$$\begin{array}{rcccl} 0 & \le & j_1,\, j_2 & \le & 20 \\ -j_1 & \le & i_1 & \le & 0 \\ -10 & \le & i_1 & & \\ -j_2 & \le & i_2 & \le & 0 \\ -10 & \le & i_2 & & \\ -j_1 - i_1 & \le & k_1 & \le & -1 \\ -10 - i_1 & \le & k_1 & & \\ -j_2 - i_2 & \le & k_2 & \le & -1 \\ -10 - i_2 & \le & k_2 & & \end{array}$$

We write the equations as usual as

$$\begin{pmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ j_1 \\ j_2 \\ k_1 \\ k_2 \\ \Delta i \\ \Delta j \\ \Delta k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Here is the GCD reduction:

$$\mathbf{A} \qquad \qquad \mathbf{I}_9$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange the first two columns:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 1 to column 5:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Subtract column 2 from column 3; add it to column 4:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 3 to columns 4 and 7; subtract it from column 5:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 4 and 5 (*and don't touch columns 7, 8, or 9*):

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 4 to columns 7 and 8:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & -1 & -1 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Interchange columns 5 and 6:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & -1 & -1 & -1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Add column 5 to columns 7, 8, and 9:

$$\underset{\mathbf{D}}{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}} \qquad \underset{\mathbf{U}}{\begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}$$

Now

$$\mathbf{D}\vec{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

yields

$$\vec{t} = \langle 0, 0, 0, 0, 0, t_6, t_7, t_8, t_9 \rangle$$

$$\vec{v} = \mathbf{U}\vec{t}$$

then gives us

$$
\begin{aligned}
i_1 &= t_8 \\
i_2 &= t_7 + t_8 \\
j_1 &= t_6 - t_8 \\
j_2 &= t_6 \\
k_1 &= t_7 + t_8 \\
k_2 &= t_7 + t_8 + t_9 \\
\Delta i &= t_7 \\
\Delta j &= t_8 \\
\Delta k &= t_9
\end{aligned}
$$

and the constraints are then expressed by the non-negative expressions

$t_6 - t_8$	(1)	
$-t_6 + t_8 + 20$	(2)	
t_6	(3)	
$-t_6 + 20$	(4)	
t_6		same as 3
$-t_8$	(5)	
$t_8 + 10$	(6)	
$t_6 + t_7 + t_8$	(7)	
$-t_7 - t_8$		subsumed by 9
$t_7 + t_8 + 10$	(8)	
$t_6 + t_7 + t_8$		same as 7
$-t_7 - t_8 - 1$	(9)	
$t_7 + 2t_8 + 10$	(10)	
$t_6 + 2t_7 + 2t_8 + t_9$	(11)	
$-t_7 - t_8 - t_9 - 1$	(12)	
$2t_7 + 2t_8 + t_9 + 10$	(13)	

We will call these expressions the *original constraint expressions*, since we will come back to them below.

9.3.3 Eliminating down to t_8

Since we only really care about t_7, t_8, and t_9, and since all the coefficients of t_6 are ± 1, we eliminate t_6. The expressions with positive coefficients are 1, 3, 7, and 11; those with negative coefficients are 2 and 4. Pairing them up, we have

$(1,2)$	20		
$(1,4)$	$-t_8+20$		subsumed by 5
$(3,2)$	t_8+20		subsumed by 6
$(3,4)$	20		
$(7,2)$	t_7+2t_8+20	(14)	
$(7,4)$	t_7+t_8+20		subsumed by 8
$(11,2)$	$2t_7+3t_8+t_9+20$	(15)	
$(11,4)$	$2t_7+2t_8+t_9+20$		subsumed by 13

This leaves us with expressions 5, 6, 9, 10, 11, 13, 14, 15, and 16. Let us make a table showing the expressions with positive and negative coefficients for each variable:

	$+$	$-$
t_7	$8,10,13,14,15$	$9,12$
t_8	$6,8,10,13,14,15$	$5,9,12$
t_9	$13,15$	12

Notice that although some of the coefficients in these expressions are not ± 1, the offending ones are all positive. Therefore, any elimination we make is guaranteed to be exact. In this case, the table shows us that the cheapest variable to eliminate is 9, so that is what we do next:

$(13,12)$	t_7+t_8+9	(16)	
$(15,12)$	t_7+2t_8+19		subsumed by 10

This leaves us with the following situation:

	$+$	$-$
t_7	$10,14,16$	9
t_8	$6,10,14,16$	$5,9$

It is now cheapest to eliminate t_7:

$(10,9)$	t_8+9	(17)	
$(14,9)$	t_8+19		subsumed by 17
$(16,9)$	8		

This leaves us with the following:

	$+$	$-$
t_8	17	5

These two expressions yield (on addition) the constant 9, which is ≥ 0. Therefore there is a dependence. Further, we know that $t_8 \leq 0$; that is, $\Delta j \leq 0$.

There are now two possibilities: $t_8 = 0$ or $t_8 < 0$. We consider them separately

9.3.4 $t_8 = 0$

We replace t_8 by 0 in each of the original constraint expressions. This yields the expressions

t_6	$(1a)$	
$-t_6 + 20$	$(2a)$	
t_6		same as $1a$
$-t_6 + 20$		same as $2a$
0		
10		
$t_6 + t_7$	$(7a)$	
$t_7 + 10$	$(8a)$	
$-t_7 - 1$	$(9a)$	
$t_7 + 10$		same as $8a$
$t_6 + 2t_7 + t_9$	$(11a)$	
$-t_7 - t_9 - 1$	$(12a)$	
$2t_7 + t_9 + 10$	$(13a)$	

We proceed as before:

	$+$	$-$
t_6	$1a, 7a, 11a$	$2a$
t_7	$7a, 8a, 11a, 13a$	$9a, 12a$
t_9	$11a, 13a$	$12a$

Again, we want to eliminate t_6:

$(1a, 2a)$	20	
$(7a, 2a)$	$t_7 + 20$	subsumed by $8a$
$(11a, 2a)$	$2t_7 + t_9 + 20$	subsumed by $13a$

So now we have

	$+$	$-$
t_7	$8a, 13a$	$9a, 12a$
t_9	$13a$	$12a$

We eliminate t_9 next:

$(13a, 12a)$	$t_7 + 9$	$(14a)$

Now we have

	$+$	$-$
t_7	$14a$	$9a$

and so we have $t_7 \le -1$, i.e., $\Delta i < 0$. If instead of eliminating t_9 at the last step we eliminate t_7, we would throw away (14a) and proceed as follows:

$(8a, 9a)$	9		
$(8a, 12a)$	$-t_9 + 9$	$(14a)$	subsumed by $16a$
$(13a, 9a)$	$t_9 + 8$	$(15a)$	
$(13a, 12a)$	$-t_9 + 8$	$(16a)$	

This leaves us with $-8 \leq t_9(= \Delta k) \leq 8$. Thus, we have found the directions given by $\Delta j = 0$, $\Delta i < 0$, $\Delta k = *$, $\Delta l = 0$; i.e., the dependence from the left-hand array reference (reference 2) to the right-hand reference (reference 1) has the direction vector $(=, <, *, =)$.

9.3.5 $t_8 < 0$

We add the constraint $t_8 < 0$ to the original constraint equations. Since we always write weak inequalities, we actually add the constraint $t_8 \leq -1$; i.e., we add the expression $-t_8 - 1$, which subsumes the original expression 5. Thus we have

$$\begin{array}{ll}
t_6 - t_8 & (1b) \\
-t_6 + t_8 + 20 & (2b) \\
t_6 & (3b) \\
-t_6 + 20 & (4b) \\
-t_8 - 1 & (5b) \\
t_8 + 10 & (6b) \\
t_6 + t_7 + t_8 & (7b) \\
t_7 + t_8 + 10 & (8b) \\
-t_7 - t_8 - 1 & (9b) \\
t_7 + 2t_8 + 10 & (10b) \\
t_6 + 2t_7 + 2t_8 + t_9 & (11b) \\
-t_7 - t_8 - t_9 - 1 & (12b) \\
2t_7 + 2t_8 + t_9 + 10 & (13b)
\end{array}$$

Our configuration is

	$+$	$-$
t_6	$1b, 3b, 7b, 11b$	$2b, 4b$
t_7	$7b, 8b, 10b, 11b, 13b$	$9b, 12b$
t_8	$2b, 6b, 7b, 8b, 10b, 11b, 13b$	$1b, 5b, 9b, 12b$
t_9	$11b, 13b$	$12b$

and we eliminate t_6. This amounts to the same computation as last time (since (5b) does not involve t_6). So expressions 1b, 2b, 3b, 4b, 7b, and 11b are deleted, and two new expressions are inserted:

$$\begin{array}{lll}
(7b, 2b) & t_7 + 2t_8 + 20 & (14b) \\
(11b, 2b) & 2t_7 + 3t_8 + t_9 + 20 & (15b)
\end{array}$$

Now we have

	$+$	$-$
t_7	$8b, 10b, 13b, 14b, 15b$	$9b, 12b$
t_8	$6b, 8b, 10b, 13b, 14b, 15b$	$5b, 9b, 12b$
t_9	$13b, 15b$	$12b$

Next, we eliminate t_8. We do this here because we already know that $t_8 < 0$; normally we would not eliminate t_8 at this stage because it would be the most expensive elimination.

$$\begin{array}{llll}
(6b, 5b) & 9 & & \\
(6b, 9b) & -t_7 + 9 & (16b) & \text{subsumed by } 21b
\end{array}$$

$(6b, 12b)$	$-t_7 - t_9 + 9$	$(17b)$	
$(8b, 5b)$	$t_7 + 9$	$(18b)$	subsumed by $20b$
$(8b, 9b)$	9		
$(8b, 12b)$	$-t_9 + 9$	$(19b)$	subsumed by $25b$
$(10b, 5b)$	$t_7 + 8$	$(20b)$	
$(10b, 9b)$	$-t_7 + 8$	$(21b)$	
$(10b, 12b)$	$-t_7 - 2t_9 + 8$	$(22b)$	
$(13b, 5b)$	$2t_7 + t_9 + 8$	$(23b)$	
$(13b, 9b)$	$t_9 + 8$	$(24b)$	
$(13b, 12b)$	$-t_9 + 8$	$(25b)$	(subsumed by $27b$)
$(14b, 5b)$	$t_7 + 18$		subsumed by $20b$
$(14b, 9b)$	$-t_7 + 18$		subsumed by $21b$
$(14b, 12b)$	$-t_7 - 2t_9 + 18$		subsumed by $22b$
$(15b, 5b)$	$2t_7 + t_9 + 17$		subsumed by $23b$
$(15b, 9b)$	$-t_7 + t_9 + 17$	$(26b)$	
$(15b, 12b)$	$-t_7 - 2t_9 + 17$		subsumed by $22b$

and so we have this configuration:

	$+$	$-$
t_7	$20b, 23b$	$17b, 21b, 22b, 26b$
t_9	$23b, 24b, 26b$	$17b, 22b, 25b$

We now eliminate t_7:

$(20b, 17b)$	$-t_9 + 17$		$(27b)$	subsumed by $28b$
$(20b, 21b)$	16			
$(20b, 22b)$	$-2t_9 + 16$	$\Longrightarrow -t_9 + 8$		subsumed by $27b$
$(20b, 26b)$	$t_9 + 25$			subsumed by $24b$
$(23b, 17b)$	$-t_9 + 26$			subsumed by $27b$
$(23b, 21b)$	$t_9 + 24$			subsumed by $24b$
$(23b, 22b)$	$-3t_9 + 24$	$\Longrightarrow -t_9 + 8$	$(28b)$	
$(23b, 26b)$	$3t_9 + 42$	$\Longrightarrow t_9 + 14$		subsumed by $24b$

This leaves us with 24b and 28b. These yield $-8 \le t_9 \le 8$. We have to consider these three cases separately:

$t_9 < 0$

We add the expression $-t_9 - 1$ to expressions 16b through 26b. It subsumes 25b, and we have

$-t_7 - t_9 + 9$	$(17c)$	
$t_7 + 8$	$(20c)$	(subsumed by $27c$)
$-t_7 + 8$	$(21c)$	
$-t_7 - 2t_9 + 8$	$(22c)$	
$2t_7 + t_9 + 8$	$(23c)$	

$t_9 + 8$	$(24c)$
$-t_9 - 1$	$(25c)$
$-t_7 + t_9 + 17$	$(26c)$

We have

	$+$	$-$
t_7	$20c, 23c$	$17c, 21c, 22c, 26c$
t_9	$23c, 24c, 26c$	$17c, 22c, 25c$

and we eliminate t_9:

$(23c, 17c)$	$t_7 + 17$			subsumed by $20c$
$(23c, 22c)$	$3t_7 + 24$	$\Longrightarrow t_7 + 8$		same as $20c$
$(23c, 25c)$	$2t_7 + 7$	$\Longrightarrow t_7 + 3$	$(27c)$	
$(24c, 17c)$	$-t_7 + 17$			subsumed by $21c$
$(24c, 22c)$	$-t_7 + 24$			subsumed by $21c$
$(24c, 25c)$	7			
$(26c, 17c)$	$-2t_7 + 26$	$\Longrightarrow -t_7 + 13$		subsumed by $21c$
$(26c, 22c)$	$-3t_7 + 42$	$\Longrightarrow -t_7 + 14$		subsumed by $21c$
$(26c, 25c)$	$-t_7 + 16$			subsumed by $21c$

which leaves us with 21c and 27c: $-3 \leq t_7 \leq 8$. Thus, we have the "direction vector" $t_8 < 0, t_9 < 0, t_7 = *$.

$t_9 = 0$

We set t_9 to 0 in the expressions 16b through 26b. This yields

$-t_7 + 9$		$(17c)$	subsumed by $20b$
$t_7 + 8$		$(20b)$	subsumed by $23b$
$-t_7 + 8$		$(21b)$	
$-t_7 + 8$		$(22b)$	same as $20b$
$2t_7 + 8$	$\Longrightarrow t_7 + 4$	$(23b)$	
8		$(24b)$	trivial
-1		$(24b)$	inconsistent
$-t_7 + 17$		$(26b)$	subsumed by 23

This yields the contradiction in 24b. So $t_9 = 0$ is not possible in this context.

$t_9 > 0$

We add the expression $t_9 - 1$ to expressions 16b through 26b. This gives

$-t_7 - t_9 + 9$	$(17c)$	
$t_7 + 8$	$(20c)$	
$-t_7 + 8$	$(21c)$	(subsumed by $28c$)
$-t_7 - 2t_9 + 8$	$(22c)$	
$2t_7 + t_9 + 8$	$(23c)$	
$t_9 + 8$	$(24c)$	
$-t_9 + 8$	$(25c)$	
$-t_7 + t_9 + 17$	$(26c)$	
$t_9 - 1$	$(27c)$	

This gives us the configuration

	$+$	$-$
t_7	$20c, 23c$	$17c, 21c, 22c, 26c$
t_9	$23c, 24c, 26c, 27c$	$17c, 22c, 25c$

and we eliminate t_9. Many of the eliminations are the same as above when we handled the case $t_9 < 0$; this can be implemented efficiently by remembering these computations. We get

$(23c, 17c)$	$t_7 + 17$			subsumed by $20c$
$(23c, 22c)$	$3t_7 + 24$	$\Longrightarrow t_7 + 8$		same as $20c$
$(23c, 25c)$	$2t_7 + 16$	$\Longrightarrow t_7 + 8$		same as $20c$
$(24c, 17c)$	$-t_7 + 17$			subsumed by $21c$
$(24c, 22c)$	$-t_7 + 24$			subsumed by $21c$
$(24c, 25c)$	16			
$(26c, 17c)$	$-2t_7 + 26$	$\Longrightarrow -t_7 + 13$		subsumed by $21c$
$(26c, 22c)$	$-3t_7 + 42$	$\Longrightarrow -t_7 + 14$		subsumed by $21c$
$(26c, 25c)$	$-t_7 + 25$			subsumed by $21c$
$(27c, 17c)$	$-t_7 + 8$			same as $21c$
$(27c, 22c)$	$-t_7 + 6$		$(28c)$	
$(27c, 25c)$	7			

This leaves 20c and 28c; i.e., $-8 \le t_7 \le 6$. So we have the "direction vector" $(t_8 < 0, t_9 > 0, t_7 = *)$

And so finally, we see that for the dependence from the left-hand reference to the right-hande reference we have the complete set of direction vectors (in the order (j, i, k, l) of the loop nest):

$(t_8 = 0, t_7 < 0, t_9 = *)$	$(=, <, *, =)$
$(t_8 < 0, t_7 = *, t_9 < 0)$	$(<, *, <, =)$
$(t_8 < 0, t_7 = *, t_9 = 0)$	$(<, *, =, =)$
$(t_8 < 0, t_7 = *, t_9 > 0)$	$(<, *, >, =)$

which could also be written more compactly as the two direction vectors

$$(-, <, *, =) \quad \text{and} \quad (<, *, *, =)$$

References

[1] Utpal Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Boston, 1988. [QA76.5.B264].

[2] Michael Burke and Ron Cytron. Interprocedural dependence analysis and parallelization. In *ACM SIGPLAN '86 Symposium on Compiler Construction, held in Palo Alto, California, June 1986*, 1986. Published in SIGPLAN Notices, Volume 21, Number 7, 1986.

[3] Gina Goff, Ken Kennedy, and Chau-Wen Tseng. Practical dependence testing. In *SIGPLAN Conference on Programming Language Design and Implementation*, Toronto, Canada, 1991. ACM.

[4] Donald Knuth. *Seminumerical Algorithms: The Art of Computer Programming, Volume 2.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981. [QA76.6.K64].

[5] Dror Eliezer Maydan. *Accurate Analysis of Array References.* PhD thesis, Stanford University, September 1992. Available at `http://suif.stanford.edu/papers/maydan92c.ps`.

[6] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, August 1992.

[7] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, England, 1986. [T57.74.S53].

[8] Michael Joseph Wolfe. *High Performance Compilers for Parallel Computing.* Addison-Wesley, Redwood City, California, 1995. [QA76.58.W62].

[9] David Wonnacott. *Constraint Based Array Dependence Analysis.* PhD thesis, University of Maryland, August 1995. Available at `ftp://ftp.cs.umd.edu/pub/omega/davewThesis`.

[10] Hans Zima. *Supercompilers for Parallel and Vector Computers.* ACM Press (New York) and Addison-Wesley (Wokingham, England and Reading, Massachusetts), 1990. with Barbara Chapman [QA76.6.Z56].