Dependence Testing

Main Theme

- The Problem: Determining whether dependences exist between two subscripted references to the same array in a loop nest
- The Solution: Learn about several different tests to detect these dependences
  - These different tests will go from simple tests to difficult tests
The General Problem

\begin{align*}
\text{DO } i_1 &= L_1, U_1 \\
\text{DO } i_2 &= L_2, U_2 \\
&\quad \vdots \\
\text{DO } i_n &= L_n, U_n \\
S_1 &\quad A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \\
S_2 &\quad \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n)) \\
\text{ENDDO} \\
&\quad \ldots \\
\text{ENDDO} \\
\text{ENDDO}
\end{align*}

\[ f_i(\alpha) = g_i(\beta) \text{ for all } i, 1 \leq i \leq m \]

The above equation is based on the observation that two array accesses are to the same memory location if and only if each corresponding subscript entry is identical. Otherwise the two references are independent.

Dependence Testing

\begin{itemize}
\item There are two goals for dependence testing
  \begin{enumerate}
  \item Prove that no dependence exists between given pairs of subscripted references to the same array variable. This is the desired outcome
    \begin{itemize}
    \item Show there is no solution in the region of appropriate $\alpha$ and $\beta$
    \end{itemize}
  \item Characterize the possible dependences in some manner, usually as a minimal complete set of distance and direction vectors
  \end{enumerate}
\item Testing must be conservative, that is, it must assume the existence of any possible dependence whose existence it cannot explicitly disprove
\end{itemize}
Background and Terminology

- **Background:**
  - Distance vectors
  - Direction vectors
  - Their application to data dependence

- **Goal of dependence testing is to construct the complete set of distance and direction vectors representing potential dependences between an arbitrary pair of subscripted references to the same array variable.**

- **Since distance vectors may be treated as precise direction vectors, only direction vectors are used.**

Basics: Indices and Subscripts

- **Index:** Index variable for some loop surrounding a pair of references

- **Subscript:** A pair of subscript positions in a pair of array references

- **For Example:**
  \[
  A(i,j) = A(i,k) + C
  \]
  - \(i, i\) is the first subscript
  - \(j, k\) is the second subscript
Basics: Linearity

- General polynomials are too complex to permit solutions to the equation $f_i(\alpha) = g_i(\beta)$ for all $i$, $1 \leq i \leq m$
- Thus, we assume all subscript expressions are of the form
  \[ a_1i_1 + a_2i_2 + \ldots + a_ni_n + e \]
- Equations of this form are said to be linear
- Any subscript that does not conform to this restriction is classified as a nonlinear subscript and is not tested

Basics: Conservative Testing

- A Diophantine equation is an equation in which only integer solutions are allowed
- The problem of finding integer solutions to system of linear Diophantine Equations is $NP$-complete
- Most common approximation is conservative testing
  - See if you can assert
    "No dependence exists between two subscripted references of the same array"
- If a conservative test determines that no dependence exists, a compiler can rely on that conclusion
- However, it may be the case that the references are independent, but the conservative test is unable to prove it
- Exact tests are dependence tests that detect dependences if and only if they exist
Basics: Complexity

- Complexity refers to the number of indices appearing with a subscript - the more distinct loop indices that appear within a subscript position, the more complex dependence testing becomes
- A subscript is said to be
  - ZIV if it contains no index
  - SIV if it contains only one index
  - MIV if it contains more than one index
- For Example:
  
  ```
  DO i
    DO j
      DO k
        A(5,i+1,j) = A(N,i,k) + C
      ENDDO
    ENDDO
  ENDDO
  ```

  First subscript is ZIV
  Second subscript is SIV
  Third subscript is MIV

Basics: Separability

- Separability describes whether a given subscript interacts with other subscripts for the purpose of dependence testing
- A subscript is separable if its indices do not occur in other subscripts
- If two different subscripts contain the same index they are coupled
- For Example:
  
  ```
  - A(I+1,J) = A(K,J) + C
    - Both subscripts are separable
  - A(I,J,J) = A(I,J,K) + C
    - Second and third subscripts are coupled
  ```
Basics: Coupled Subscript Groups

- Why are they important?
- Coupling can cause imprecision in dependence testing

```
DO I = 1, 100
  S1 A(I+1,I) = B(I) + C
  S2 D(I) = A(I,I) * E
ENDDO
```

— First subscript in S1 equals the first subscript in S2 one iteration later
— Second subscript in S1 equals the second subscript in S2 on the same iteration
— Alone, neither subscript can be used to eliminate the dependence
— Together, no dependence exists since the dependence cannot be simultaneously loop carried and loop independent

Dependence Testing: Overview

1. Partition the subscripts into separable and minimal coupled groups
2. Classify each subscript as ZIV, SIV, MIV
3. For each separable subscript, apply the appropriate single subscript test (ZIV, SIV, MIV), based on the complexity of the subscript, to either prove independence or produce direction vectors for the indices occurring in that subscript. If independence is proved, no further testing is needed
4. For each coupled group, apply a multiple subscript test to produce a set of direction vectors for the indices occurring within that group
5. If any test yields independence, no further testing is needed because no dependence exists
6. Otherwise merge all direction vectors computed in the previous steps into a single set of direction vectors for the two references
Subscript Partitioning

- Partitions the subscripts into separable and minimal coupled groups
- The partition algorithm initially places each subscript pair in its own partition, then merges together partitions that refer to the same loop variable
- Notations
  // S is a set of m subscript pairs S_1, S_2, ... S_m each enclosed in
  // n loops with indexes I_1, I_2, ... I_n which is to be
  // partitioned into separable or minimal coupled groups.
  // P is an output variable, containing the set of partitions
  // n_p is the number of partitions

Subscript Partitioning Algorithm

procedure partition(S, P, n_p)
    n_p = m;
    for i = 1 to m do
        P_i ← {S_i};
    for i = 1 to n do begin
        k ← <none>
        for each remaining partition P_j do
            if there exists s ∈ P_j such that s contains I_i then
                if k = < none > then k ← j;
                else begin P_k ← P_k ∪ P_j; discard P_j; n_p ← n_p - 1; end
        end
    end
end partition
Classify as ZIV/SIV/MIV

- Easy step
- Just count the number of different indices in a subscript

Apply Single Subscript Tests

- ZIV test
  - If the two expressions can be proved not equal, then the corresponding array references are independent
  - If the expressions cannot be shown to be different, then the subscript does not contribute to any direction vector and may be ignored

- SIV test
  - Strong SIV test
  - Weak SIV test
    - Weak-zero SIV
    - Weak crossing SIV

- SIV tests in complex iteration spaces
Merging Direction Vectors

- Consider
  DO I
    DO J
      A(I+1,J) = A(I,J) + C
    ENDDO
  ENDDO
  - 1st subscript yields (§)
  - 2nd subscript yields (=)
  - Merged subscript yields (§,=)

- Consider
  DO I
    DO J
      A(I+1,5) = A(I,N) + C
    ENDDO
  ENDDO
  - 1st subscript yields (§)
  - 2nd has no variance
  - Merged subscript yields (§,§), (§,=), (§,>)

ZIV Test

DO J = 1, 100
S  A(e_1) = A(e_2) + B(J)
ENDDO

- e_1, e_2 are constants or loop invariant symbols
- If (e_1-e_2) ≠ 0, then no dependence exists
Strong SIV Test

- Strong SIV subscripts are of the form
  \[ \langle ai + c_1, ai' + c_2 \rangle \]

- For example, the following are strong SIV subscripts
  \[ \langle i +1, i \rangle \]
  \[ \langle 4i +2, 4i +4 \rangle \]

**Dependence distance**

\[ d = i' - i = \frac{c_1 - c_2}{a} \]

**Dependence exists if**

\[ |d| \leq U - L \]
Strong SIV Test Example

\begin{verbatim}
DO I = 1, N
S1 A(I+2*N) = A(I+N) + C
ENDDO

\end{verbatim}

\begin{itemize}
\item d = 2N - N
\item Independent since N > N - 1
\end{itemize}

Weak SIV Tests

\begin{itemize}
\item Weak SIV subscripts are of the form
\[ a_i + c_1 = a_{i'} + c_2 \]
\item For example the following are weak SIV subscripts
\[ \langle i + 1, 5 \rangle \]
\[ \langle 2i + 1, i + 5 \rangle \]
\[ \langle 2i + 1, -2i \rangle \]
\item The dependence equation is: \( a_1 i + c_1 = a_2 i' + c_2 \)
\end{itemize}
Weak SIV Tests

A special case of Weak SIV is when one of the coefficients of the index is zero.

\[ i = \frac{c_2 - c_1}{a_1} \]

- The test consists merely of checking whether the solution is an integer and is within loop bounds.

Weak-Zero SIV Test
Weak-Zero SIV Test

Weak-Zero SIV & Loop Peeling

DO I = 1, N
S1  Y(I, N) = Y(1, N) + Y(N, N)
ENDDO

Where is the dependence?

Can be loop peeled to...

Y(1, N) = Y(1, N) + Y(N, N)
DO I = 2, N-1
S1  Y(I, N) = Y(1, N) + Y(N, N)
ENDDO
Y(N, N) = Y(1, N) + Y(N, N)
Weak-Crossing SIV Test

- Special case of Weak SIV where the coefficients of the index are equal in magnitude but opposite in sign
  \[ a_2 = -a_1 \]
- The test consists merely of checking whether the solution index is
  1. Within loop bounds and is
  2. Either an integer or has a non-integer part equal to 1/2

\[ i = \frac{c_2 - c_1}{2a_1} \]
Weak-crossing SIV & Loop Splitting

\[
\text{DO } I = 1, N \\
S_1 \quad A(I) = A(N-I+1) + C \\
\text{ENDDO}
\]

This loop can be split into...

\[
\text{DO } I = 1, (N+1)/2 \\
S_1 \quad A(I) = A(N-I+1) + C \\
\text{ENDDO} \\
\text{DO } I = (N+1)/2 + 1, N \\
S_1 \quad A(I) = A(N-I+1) + C \\
\text{ENDDO}
\]

Complex Iteration Spaces

• Until now we have applied the tests only to rectangular iteration spaces
• These tests can also be extended to apply to triangular or trapezoidal loops:
  — Triangular: One of the loop bounds is a function of at least one other loop index
  — Trapezoidal: Both the loop bounds are functions of at least one other loop index
Complex Iteration Spaces

• For example consider this special case of a strong SIV subscript

\[
\text{DO } I = 1, N \\
\text{DO } J = L_0 + L_1 \ast I, U_0 + U_1 \ast I \\
S_1 \quad A(J + D) = \ldots \\
S_2 \quad \ldots = A(J) + B \\
\text{ENDDO} \\
\text{ENDDO}
\]

• By the strong SIV test, we can see that there is a dependence of statement \( S_2 \) on statement \( S_1 \) carried by the \( J \)-loop if the dependence distance \( d \) is smaller in absolute value than the distance from the loop lower bound to the loop upper bound

\[
|d| \leq U_0 - L_0 + |U_1 - L_1| I \\
I \geq \frac{|d| - |U_0 - L_0|}{U_1 - L_1}
\]

• Unless this inequality is violated for all values of \( I \) in its iteration range, we must assume a dependence in the loop
Index Set Splitting

\[
\begin{align*}
\text{DO } & I = 1, 100 \\
\text{DO } & J = 1, I \\
S_1 & \quad A(J + 20) = A(J) + B \\
\text{ENDDO} & \\
\text{ENDDO} & \\
\text{DO } & I = 1, N \\
\text{DO } & J = L_0 + L_1 * I, U_0 + U_1 * I \\
S_1 & \quad A(J + D) = \ldots \\
S_2 & \quad \ldots = A(J) + B \\
\text{ENDDO} & \\
\text{ENDDO} & \\
\end{align*}
\]

• For values of

\[
I < \frac{|A-(U_0-L_0)}{U_1-L_1} = \frac{20-(-1)}{1} = 21
\]

there is no dependence

• This is called the breaking condition
  —The condition(s) under which the dependence does not hold

Index Set Splitting

• This condition can be used to partially vectorize \( S_1 \) by index set splitting as shown

\[
\begin{align*}
\text{DO } & I = 1, 20 \\
\text{DO } & J = 1, I \\
S_{1a} & \quad A(J + 20) = A(J) + B \\
\text{ENDDO} & \\
\text{ENDDO} & \\
\text{DO } & I = 21, 100 \\
\text{DO } & J = 1, I \\
S_{1b} & \quad A(J + 20) = A(J) + B \\
\text{ENDDO} & \\
\text{ENDDO} & \\
\end{align*}
\]

• Now the inner loop of the first nest can be vectorized
**Coupling makes these tests imprecise**

\[
\begin{align*}
&\text{DO } I = 1, 100 \\
&\quad \text{DO } J = 1, I \\
&\quad S_1 \quad A(J + 20, I) = A(J, 19) + B \\
&\quad \text{ENDDO} \\
&\text{ENDDO}
\end{align*}
\]

- **Dependence**
  - With strong SIV test on first subscript, no dependence unless \( I \geq 21 \)
  - With weak SIV test on second subscript, dependence only when \( I = 19 \)

- A dependence is reported if the two subscripts are tested separately
- But since they are coupled, no dependence is reported

---

**Breaking Conditions**

- **Consider the following example**
  
  \[
  \begin{align*}
  &\text{DO } I = 1, L \\
  &S_1 \quad A(I + N) = A(I) + B \\
  &\text{ENDDO}
  \end{align*}
  \]

- If \( L \leq N \), then there is no dependence from \( S_1 \) to itself
- \( L \leq N \) is called the breaking condition
Using Breaking Conditions

- Using breaking conditions, the vectorizer can generate alternative code

  IF (L <= N) THEN
      A(N+1:N+L) = A(1:L) + B
  ELSE
      DO I = 1, L
          S1 A(I + N) = A(I) + B
      ENDDO
  ENDIF

Another Example

  DO I = 1, N
      S1 A(I) = A(L) + B
  ENDDO

- Assume no relationship between N and L
- Weak-zero test cannot prove there is no dependence
- Breaking condition: (L < 1) .OR. (L > N)
- Vectorizer may generate code such as:

  IF (( L.LT. 1 ) .OR. (L.GT. N )) THEN
      A(1:N) = A(L) + B
  ELSE
      DO I = 1, N
          S1 A(I) = A(L) + B
      ENDDO
  ENDIF
Recall...

- **General Dependence:**
  
  Let \((D_1, D_2, ..., D_n)\) be a direction vector, and consider the following loop nest
  
  \[
  \begin{align*}
  &\text{DO } i_1 = L_1, U_1 \\
  &\quad \text{DO } i_2 = L_2, U_2 \\
  &\quad \quad \vdots \\
  &\quad \quad \text{DO } i_n = L_n, U_n \\
  &\quad S_1 \quad f(i) = \ldots \\
  &\quad S_2 \quad \ldots = g(i) \\
  &\quad \text{ENDDO} \\
  \end{align*}
  \]

  Then \(S_2 \delta S_1\) if \(f(x) = g(y)\) can be solved for iteration vectors \(x, y\) that agree with \(D\).

Recall...

- Determining whether there is a dependence with direction vector \(D\) is equivalent to determining whether there exists an integer solution to the equation system

  \[
  f(x_1, x_2, ..., x_n) = g(y_1, y_2, ..., y_n)
  \]

  in the space defined by

  \[
  L_i \leq x_i, y_i \leq U_i, \forall i, 1 \leq i \leq n
  \]

  with the additional restriction due to the direction vector:

  \[
  x_i D_1 y_i, \forall i, 1 \leq i \leq n
  \]
Recall...

- Last time we cared about cases where \( f \) and \( g \) each involved a single induction variable.
- There were several special cases that helped matters:
  - Strong SIV
  - Weak-zero SIV
  - Weak-crossing SIV

The General Case

- We must relax our restrictions on \( f \) and \( g \) to let them be arbitrary functions.
- A dependency exists if

\[
\frac{1}{h(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m)} = f(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m) - g(x_1, x_2, ..., x_n, y_1, y_2, ..., y_m) = 0
\]

has an integer solution

- The above equation is known as a Diophantine equation
Diophantine Equations

• Functional equations with interest focused on integer solutions
• Example: \( x^2 + y^2 = z^2 \) is the Pythagorean equation.
  \(- (l,m,n) \) is a solution with \( \gcd(l,m) = 1 \) iff we have integers \( a \) and \( b \),
  \( \gcd(a,b) = 1 \) with:
  \- \( a + b \) an odd integer
  \- \( l = 2a \)
  \- \( m = a^2 - b^2 \)
  \- \( n = a^2 + b^2 \)

Linear Diophantine Equations

• For simplicity, assume that
  \[ f(x) = a_0 x_0 + a_1 x_1 + ... + a_n x_n \]
  \[ g(x) = b_0 y_0 + b_1 y_1 + ... + b_n y_n \]
• Then, we're looking for a solution of
  \[ h(x) = a_0 - b_0 + a_1 x_1 - b_1 y_1 + ... + a_n x_n - b_n y_n = 0 \]
• Rearranging terms, we get the linear Diophantine Equation:
  \[ a_0 x_0 - b_0 y_0 + ... + a_n x_n - b_n y_n = b_o - a_o \]
Linear Diophantine Equations

- A basic result tells us that there are values for $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$ so that
  \[ a_1 x_1 - b_1 y_1 + \ldots + a_n x_n - b_n y_n = \gcd(a_1, \ldots, a_n, b_1, \ldots, b_n) \]

  What’s more, $\gcd(a_1, \ldots, a_n, b_1, \ldots, b_n)$ is the smallest number this is true for (group theory in disguise)

- As a result, the equation has a solution iff $\gcd(a_1, \ldots, a_n, b_1, \ldots, b_n)$ divides $b_0 - a_0$

- Thus, if the $\gcd$ of all the coefficients of loop induction variables does not divide the difference of the two constant additive terms, there can be no solution to the equation anywhere – hence, no dependence can exist.

gcd Example

- Consider the code
  ```
  DO I = 0, 10
     DO J = 0, 10
       S1 A(2*I+J) = ...
       S2 ... = A(-I+2*J-21)
     ENDDO
   ENDDO
  ```

- The $\gcd$ equation is:
  \[ \gcd(2,1,-1,2) \mid 21 \]
  and says there is a possible dependence

- However, there is no solution to the dependence equation in the region \[ \{ 0 \leq x_1, x_2, y_1, y_2 \leq 10 \} \]
Another Example

- For the code
  DO I = 1, N
    DO J = 2, M
      S_1 = A(2*I-2*J) = ...
      S_2 = A(4*I-6*J+3)
    ENDDO
  ENDDO
- We have the equation
  \( \gcd(2,-2,4,6) \mid 3 \)
- Thus there is no dependence between \( S_1 \) and \( S_2 \)

Special Case for GCD

- When testing for a specific direction vector \( \mathbf{D}=(D_1,\ldots,D_n) \), some of whose directions are \( "=" \), this condition can be tightened
- Assume that \( \mathbf{D} \) has only one \( "=" \) component, \( D_i \)
- The equation becomes
  \[ a_1x_1 - b_1y_1 + \ldots + (a_i - b_i)x_i + \ldots + a_nx_n - b_ny_n = b_0 - a_0 \]
Banerjee Test

- Although the GCD test is extremely useful in some cases, it is inadequate as a general dependence test.
- The reason is that the most common GCD encountered in practice is 1, which divides everything.
- Furthermore, the GCD test indicates dependence whenever the dependence equation has an integer solution anywhere, not just in the region R.
- One test that eliminates the above problem by considering iteration limits is the Banerjee Inequality.
- The Banerjee tests for real solutions, not just integer, under the assumption that if there is no real solution, there is no integer solution.
- Learning about the Banerjee test is left to you, if you want.

Testing for All Direction Vectors

- Must test pair of statements for all direction vectors.
- Potentially exponential in loop nesting.
- Can save time by pruning:
Coupled Groups

- So far, we’ve assumed separable subscripts
- We can glean information from separable subscripts, and use it to split coupled groups
- Most subscripts tend to be SIV, so this works pretty well.

Testing Coupled Groups

- An approach to coupled groups is to treat each subscript separately and intersect the resulting sets of direction vectors
- DO I = 1, 100
  A(I+1,I) = B(I) + C
  D(I) = A(I,I) * E
END
- The first subscript yields the direction vector (<), while the second yields (=)
- When we intersect these direction vectors, we discover that no dependence can exist
Testing Coupled Groups

- Consider the following code segment
  
  \[
  \text{DO } i = 1, 100 \\
  S_1 \quad A(i+1,i+2) = A(i,i) + c \\
  \text{ENDDO}
  \]

- Subscript-by-subscript testing gives (<,<)

- A careful examination of the statement reveals that this direction vector is invalid because no dependence actually exists - the subscript pair cannot be simultaneously equal

- To improve the above situation, we could test distance vectors instead of direction vectors and intersect the distance vectors
  - This strategy is effective because a large percentage of dependence tests applied in practice are strong SIV tests, which produce distances
  - In the above the distance vector is (1,2) and the intersection is empty, thus no dependence

Delta Test

- Used to test for dependence in multiple subscripts.
- Designed to be exact yet efficient for common coupled subscripts.
- Constraint vector \( C \) for a subscript group, contains one constraint for each index in group.
- The Delta test derives and propagates constraints from SIV subscripts.
- Constraints are also propagated from RDIV (Restricted Double Index Variable) subscripts, those of the form
  
  \(<a_i+c_1,a_j+c_2>\)
Delta Test

Procedure Delta(subscr, constr)

1. Init constraint vector $C$ to <none>

While exist untested SIV subscripts in subscr

   Apply SIV test to all untested SIV subscripts
   Return independence, or derive new constraint vector $C'$. 
   $C' \leftarrow C \cap C'$

   If $C' = \emptyset$ Then return independence
   Else If $C \neq C'$ Then
       $C \leftarrow C'$
       Propagate $C$ into MIV subscripts
       Apply ZIV test to untested ZIV subscripts
       Return independence if no solution

While exist untested RDI V subscripts

   Test and propagate RDI V constants
   Test remaining MIV subscripts using MIV tests
   Intersect direction vectors with $C$, and return

SIV Constraints

DO I

   DO J

   $S_I \quad A(I+1,I+J) = A(I,I+J)$

ENDDO

ENDDO

• First subscript gives a dependence distance of (1)
• Propagating this to second subscript and getting rid of I we get:
  — Add $-(I+1),-I$ to the second subscript
  — Result is $<J-1,J>$
• The resulting distance vector is (1,-1)
More on SIV Constraints

\begin{verbatim}
DO I
  DO J
    DO K
      S_1_ A(J-1,I+1,J+K) = A(J-1,I,J+K)
      ENDDO
    ENDDO
  ENDDO

- Second subscript has a dependence distance of (1)
- Propagate this to the first subscript
  - Add <I+1,I> resulting in <J+1,J>
  - First subscript has a dependence distance of (1)
- Propagate this to the third subscript
  - Add <-K,J+1> resulting in <K-1,K>
  - Third subscript has a dependence distance of (-1)
- Final distance vector is (1,1,-1)
\end{verbatim}

Improved Precision

\begin{verbatim}
DO I = 1, 100
  DO J = 1, 100
    S_1_ A(I-1,2*I) = A(I,I+J+10)
  ENDDO
ENDDO

- First subscript has a distance vector of (-1)
- Propagate this to second subscript
  - Add <-2(I-1),-2i> resulting in <2,J-I+110>
- This can now be handled by the Banerjee Inequality test
\end{verbatim}
Another Delta Example

DO I
   DO J
      S1 A(I,2*J+I) = A(I,2*J-I+5)
   ENDDO
• GCD shows integer solution for both subscripts
  — GCD(2,1,2,-1) divides 5
• However, propagating the distance constraint <0> for I from the first subscript into the second yields
  — Add <-I,-I> resulting in <2*J, 2*J-2*I+5>
  — The GCD test now detects independence because GCD(2,2,-2) does not divide 5
  — The subscripts are independent

Delta Test and Distance Vector

DO I = 1, N
   DO J = I+1, I+N
      A(I,J-I) = A(I-1,J-I) + c
   ENDDO
ENDDO
• First subscript gives us a distance of 1
• Propagated into the second subscript
  — Add <(I+1),I> resulting in <J+1,J>
  — Second subscript has a distance of (1)
• The resulting distance vector is (1, 1)
Delta Test

- Can detect independence if and only if any of its component SIV tests detect independence
- Otherwise, it converts all SIV subscripts into constraints, propagating them into MIV subscripts where possible
- The process repeats until no new constraints are found
- The remaining MIV subscripts are tested and the results are intersected with existing constraints

Final Assembly

- Basic dependence algorithm:
  - Figure out what sort of subscripts we have
  - Partition subscripts into coupled groups
  - For each separable subscript
    - Test it using appropriate test
    - If no dependence, we're done
  - For each coupled group
    - Use Delta test
    - If no dependence, we're done
  - Return dependence
More Techniques

- Solving $h(x) = 0$ is essentially an integer programming problem. Linear programming techniques are used.