Dependence analysis concentrates on loops, primarily since it is ultimately there where the restructuring of code may result in substantial run time savings. Traditional dependence testing concentrates on for-loops since the presence of induction variables is important for the standard techniques (see Wolfe; Zima & Chapman).

In general, while-loops are usually handled by extracting induction variables and effectively converting the while-loop into a for-loop for the purpose of dependence analysis. Unfortunately, there are while-loops that either do not possess induction variables or whose induction variables cannot be recognized as such by automatic techniques. This raises the question what to do with these (frequently semantically important) while-loops.

2. The Underlying Assumptions

We propose a very simple basic method for doing dependence analysis on arbitrary (unstructured) while-loops whose loop body is a basic block that is based on two fundamental premises:

(A) We are interested in the worst-case analysis only.

(B) Any array reference in the code is valid.

By worst-case analysis we mean that the absence of a definitive conclusion that no dependence (of statement \( S_1 \) on statement \( S_2 \)) exists is taken to be equivalent to the presence of a dependence (of \( S_1 \) on \( S_2 \)). Thus, in concordance with traditional optimization techniques we assume a dependence if our method cannot determine unequivocally that none exists.

The postulated validity of all array references is simply a consequence of the assumption that the program that is to be optimized is correct. It is an often belabored truism that only correct programs should be optimized. Thus in particular, we assume that any index into an array is within the bounds of that array (in each dimension).

Note that our basic method assumes that the body of the loop be a basic block; thus, in particular each statement will be executed exactly once in each iteration. This assumption is necessary for our basic method since otherwise a statement that is not executed (or not executed in each iteration) could incorrectly reduce the valid range of the indices. We will briefly touch upon a relaxation of this assumption at the end of this note.

3. The Basic Approach

In the following, we will use this general format of a while-loop

\[
\text{while } \text{cond } \text{do}
\]

\[
S_1;
\]

\[
S_2;
\]

\[
\ldots
\]

\[
S_n
\]

\text{od}

where each statement \( S_i \) is an assignment statement and may involve references to (multi-dimensional) arrays.

(I) Under the assumptions stated above, we first determine the valid range of each index when referencing any array. Then we obtain the valid range of each index for the entire loop by taking the intersection of the valid ranges of all references in the loop.

(II) Given the valid range of an index for a loop, we can then carry out standard dependence analysis of a loop where the index ranges over its entire valid range.

The assumption that the loop-body of our while-loop be a basic block is indispensable for the correctness of our approach. This is demonstrated by the following example where this assumption is violated:

\[
\text{while } \text{cond } \text{do}
\]

\text{first index calculation;}

\[
A(i+1) = (A(2*i) + A(N-2*i))^2 + 1.0;
\]

\text{if } A(i+1) < 1.0 \text{ then } A(i+2) = A(2*i+2) + A(2*N-3*i-2);

\text{second index calculation}

\text{od}