Examples: Data Dependence and Loop Transformation

1. Consider the following code:

```
for i = 1,N-2
  for j = i+1,i+N-2
      A(i+1,j-i+1) = A(i,j-i+1) - A(i+1,j-i)
  end for
end for
```

   (a) **Are there any output dependences? If so, what are they?**
   For an output dependence to exist between (i1,j1) to (i2,j2), the same-location conditions requires that (i1+1,j1-i1+1)=(i2+1,j2-i2+1), i.e. i1=i2, and j1-i1+1 = j2-i2+1, i.e. i1=i2 and j1=j2. So no output dependence exists.

   (b) **Are there any flow dependences? If so, what are they?**

      i. Consider write in iteration (iw,jw) into A(iw+1,jw-iw+1) and read in iteration (ir,jr) from A(ir,jr-ir+1). For a flow dependence to exist, the distance vector (ir-iw,jr-jw) must be lexicographically positive and the same location condition requires that (iw+1,jw-iw+1) =(ir,jr-ir+1). Thus iw+1=ir, i.e. ir-iw=1. Also, since jw-iw+1 must equal jr-ir+1, jr-jw = ir-iw = 1. So we have a flow dependence (1,1).

      ii. Consider write in iteration (iw,jw) into A(iw+1,jw-iw+1) and read in iteration (ir,jr) from A(ir+1,jr-ir). For a flow dependence to exist, the distance vector (ir-iw,jr-jw) must be lexicographically positive and the same location condition requires that (iw+1,jw-iw+1) =(ir+1,jr-ir). Thus iw+1=ir+1, i.e. ir-iw=0. And for jw-iw+1 to equal jr-ir, jr-jw = ir-iw+1 = 1. So we have a flow dependence (0,1).

   (c) **Are there any anti dependences? If so, what are they?**
   Consider write in iteration (iw,jw) into A(iw+1,jw-iw+1) and read in iteration (ir,jr) from A(ir,jr-ir+1). For an anti dependence to exist, the distance vector (iw-ir,jw-jr) must be lexicographically positive and the same location condition requires that (iw+1,jw-iw+1) =(ir,jr-ir+1). Thus iw+1=ir, i.e. iw-ir=-1. This means that no valid dependence vector is possible because the leftmost component cannot be -1. Therefore no anti-dependences exist. A similar analysis for the other read reference will show that no antidependence can exist for that case either.

2. Consider the following code:

```
int i,j,t;

      for (t=0;t<1024;t++)
        for (i=0;i<1024;i++)
          for (j=0;j<1024;j++)
            S(t,i,j);
```

The data dependences for the loop are given to be (1,0,-1), (0,0,1), and (1,-1,-1).

(a) **Which loops (if any) are valid to unroll? Why?**
For checking if a loop can be unrolled, the dependence vectors must be permuted corresponding to a loop permutation that makes the candidate loop innermost without changing the relative order of the other loops. If none of the permuted dependence vectors are lexicographically negative, the unrolling is legal. The middle and inner loops are valid to unroll. The outer loop is not valid to unroll since the permuted vector for (1,0,-1) is (0,-1,1), which is lex. negative, and the permuted vector for (1,-1,-1) is (-1,-1,1), which is also lex. negative.

(b) **What are the valid permutations of the loop? Why?**
The dep. vector (1,-1,-1) implies that only the t loop can stay outermost. Permuting the i and j loops does not cause any dep. vectors to become lex. negative. So tij and tji are the only valid permutations.

(c) **What tiling (if any) is valid? Why?**
Full 3D tiling requires all loops to be fully permutable. This is clearly not the case. However, partial tiling is feasible if any band of loops is fully permutable. Here, the band ij is fully permutable. Hence tiling of just the i and j loops is valid.

(d) **Which loops (if any) are parallel? Why?**
A loop is parallel if it does not carry a dependence with respect to any of the dependence vectors. With respect to a single dependence vector, a loop has no loop carried dependences if either that component of the dependence vector is zero, or a more significant position in the dep. vector (to its left) has a positive component. Considering the three loops:

- Loop t is not parallel due to the 1 in its position in the dep. vectors (1,0,-1) and (1,-1,-1).
- Loop i is parallel since it has a zero component for (1,0,-1) and (0,0,1), and although it has a -1 for (1,-1,-1), there is a one in a position to its left (corresponding to the t loop).
- Loop j is not parallel due to (0,0,1): there is a non-zero component in its position and no positive component to its left.

3. Consider the following loop, with dependences (1,0,0), (0,1,0), (0,0,1):

```
int i,j,t;

      for (t=0;t<1024;t++)
        for (i=t;i<1024;i++)
          for (j=0;j<t;j++)
            S(t,i,j);
```

(a) Show the *jit* form of the loop.
The transformed code can be generated by performing a sequence of simpler 2-loop permutations: tij to tji to jti to jit.

- tij to tji requires no change to the loop bounds of i or j since they only depend on the outer-most loop t.

- tji to jti

```
for (j=0;j<1023;j++)
  for (t=j+1;t<1024;t++)
    for (i=t;i<1024;i++)
      S(t,i,j);
```

- jti to jit

```
for (j=0;j<1023;j++)
  for (i=j+1;i<1024;t++)
    for (t=j+1;t<=i;i++)
      S(t,i,j);
```

(b) **Show a 2-way i-unrolled form (i.e., unroll-jam) of the *tij* form [Note: i loop bounds are not constant]**
There is more than one way to do this. Since the range of the i loop is odd or even depending on whether t is odd or even, it is necessary to ensure that only valid instances are evaluated. For odd values of t, full 2-way unrolling of t is not possible. One solution is:

```
for (t=0;t<1024;t++)
{ for (i=t;i<1023;i+=2)
    for (j=0;j<t;j++)
      { S(t,i,j); S(t,i+1,j);}
  if (t%2 != 0)
    for (j=0;j<t;j++) S(t,1023,j);
}
```