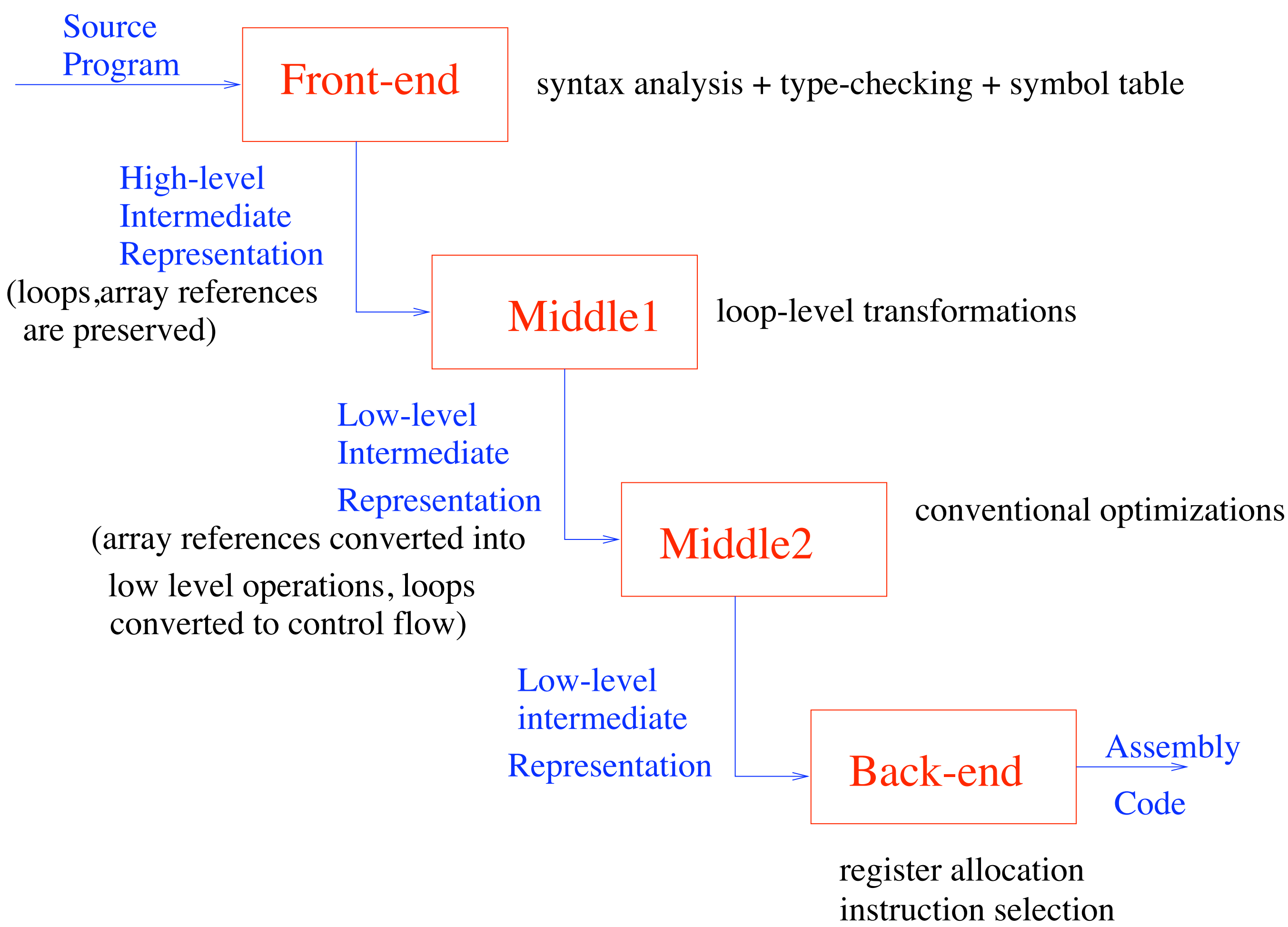
Introduction
to
Loop Transformations

Organization of a Modern Compiler



Key concepts:

Perfectly-nested loop: Loop nest in which all assignment statements occur in body of innermost loop.

```
for J = 1, N
  for I = 1, N
    Y(I) = Y(I) + A(I,J)*X(J)
```

Imperfectly-nested loop: Loop nest in which some assignment statements occur within some but not all loops of loop nest

```
  for k = 1, N
    a(k,k) = sqrt (a(k,k))
    for i = k+1, N
      a(i,k) = a(i,k) / a(k,k)
    for i = k+1, N
      for j = k+1, i
        a(i,j) -= a(i,k) * a(j,k)
```

Our focus for now: perfectly-nested loops

Goal of lecture:

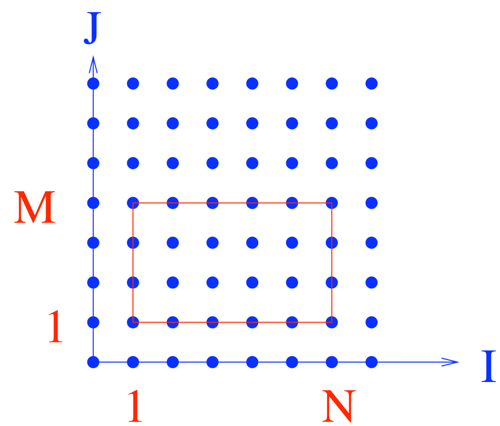
- We have seen two key transformations of perfectly-nested loops for locality enhancement: permutation and tiling.
- There are other loop transformations that we will discuss in class.
- Powerful way of thinking of perfectly-nested loop execution and transformations:
  - loop body instances $\leftrightarrow$ *iteration space* of loop
  - loop transformation $\leftrightarrow$ *change of basis* for iteration space

Iteration Space of a Perfectly-nested Loop

Each iteration of a loop nest with n loops can be viewed as an integer point in an n-dimensional space.

Iteration space of loop: all points in n-dimensional space corresponding to loop iterations

```
DO  I = 1, N
  DO  J  = 1,M
     S
```

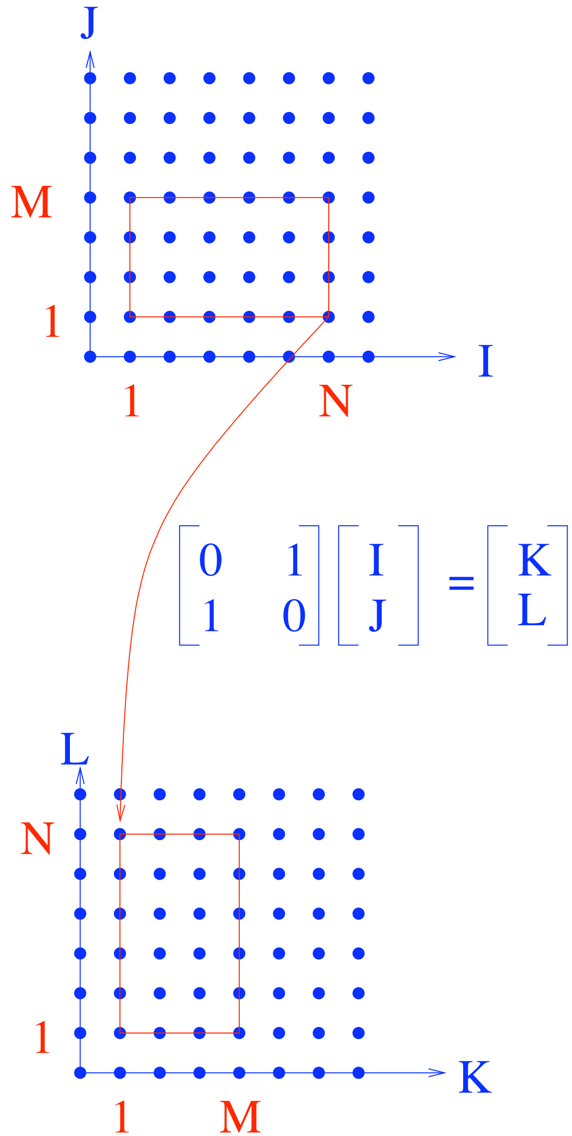


Execution order = lexicographic order on iteration space:

$(1,1) \preceq (1,2) \preceq ... \preceq (1,M) \preceq (2,1) \preceq (2,2)... \preceq (N,M)$

Loop permutation = linear transformation on iteration space

```
DO I = 1, N
  DO J = 1,M
    S
```



$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} K \\ L \end{bmatrix}$$

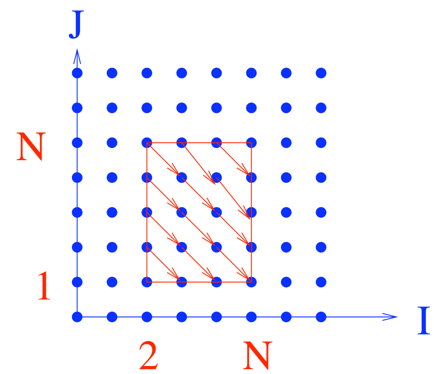
```
DO K = 1, M
  DO L = 1, N
    S
```

Locality enhancement:

Loop permutation brings iterations that touch the same cache line ”closer” together, so probability of cache hits is increased.

Subtle issue 1: loop permutation may be illegal in some loop nests

```
DO  I = 2, N
 DO  J  = 1, M
   A[I,J] = A[I-1,J+1] + 1
```



Assume that array has 1's stored everywhere before loop begins.

After loop permutation:

```
DO  J  = 1, M
  DO  I = 2, N
    A[I,J] = A[I-1,J+1] + 1
```

Transformed loop will produce different values (A[3,1] for example)
=> permutation is illegal for this loop.

Question: How do we determine when loop permutation is legal?

Subtle issue 2: generating code for transformed loop nest may be non-trivial!

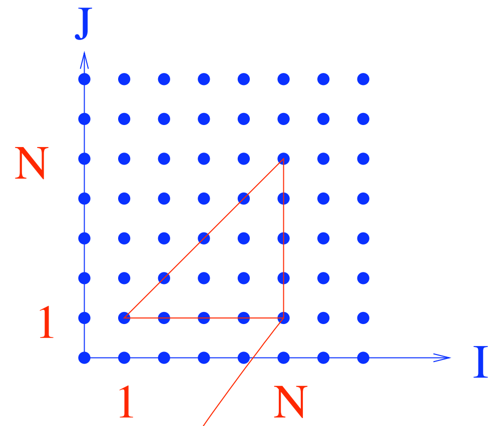
Example: triangular loop bounds (triangular solve/Cholesky)

```
FOR I = 1, N
  FOR J = 1, I-1
     S
```

Here, inner loop bounds are functions of outer loop indices!

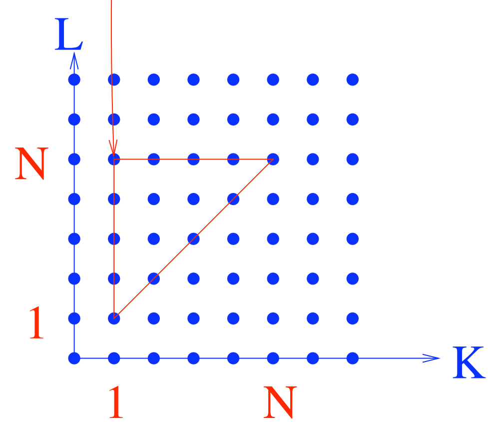
Just exchanging the two loops will not generate correct bounds.

DO I = 1, N
  DO J = 1, I
    S



$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} K \\ L \end{bmatrix}$$

DO K = 1, N
  DO L = I, N
    S



Question: How do we generate loop bounds for transformed loop nest?

General theory of loop transformations should tell us

- which transformations are legal,
- what the best sequence of transformations should be for a given target architecture, and
- what the transformed code should be.

Desirable: quantitative estimates of performance improvement

ILP Formulation
of
Loop Transformations

Goal:

1. formulate correctness of permutation as integer linear programming (ILP) problem
2. formulate code generation problem as ILP

Two problems:

Given a system of linear inequalities $A\,x \leq b$

where A is a m X n matrix of integers,

b is an m vector of integers,

x is an n vector of unknowns,

(i) Are there integer solutions?

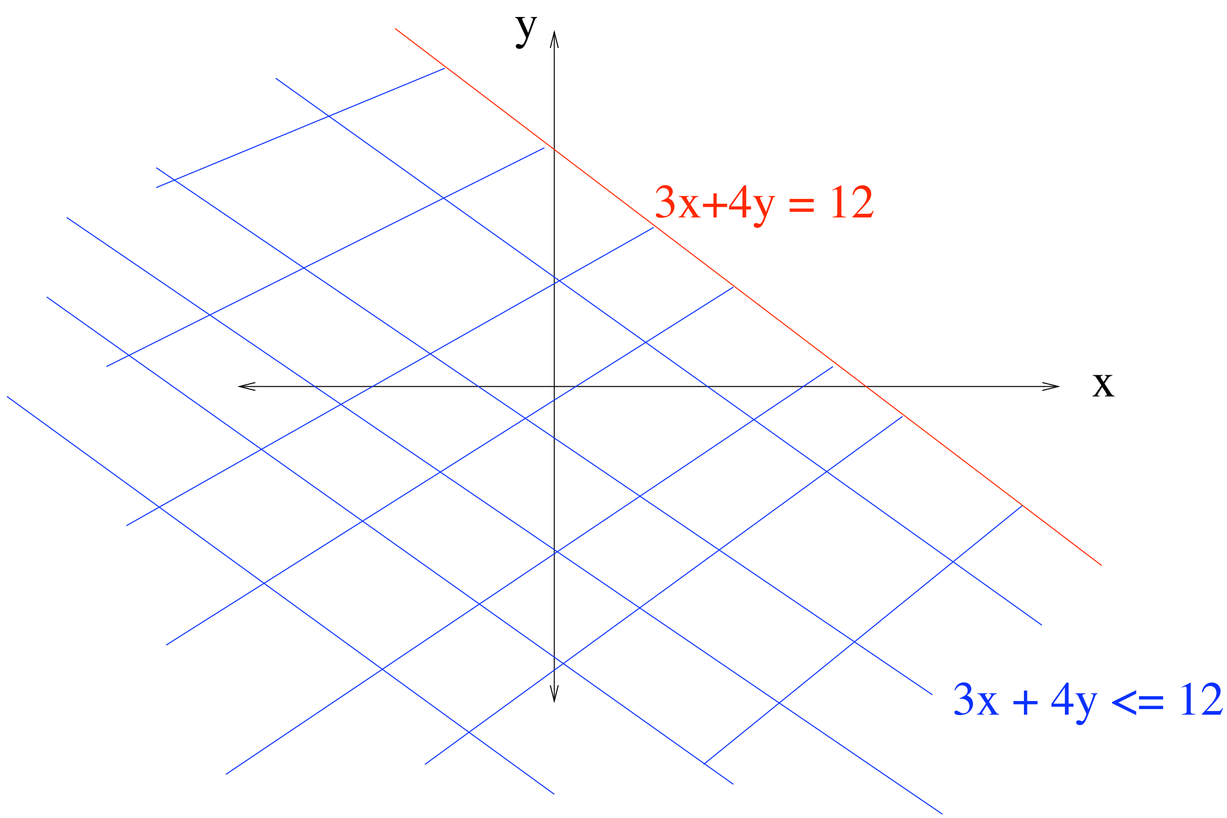
(ii) Enumerate all integer solutions.

Most problems regarding correctness of transformations and code generation can be reduced to these problems.

Intuition about systems of linear inequalities:

Equality: line (2D), plane (3D), hyperplane (> 3D)

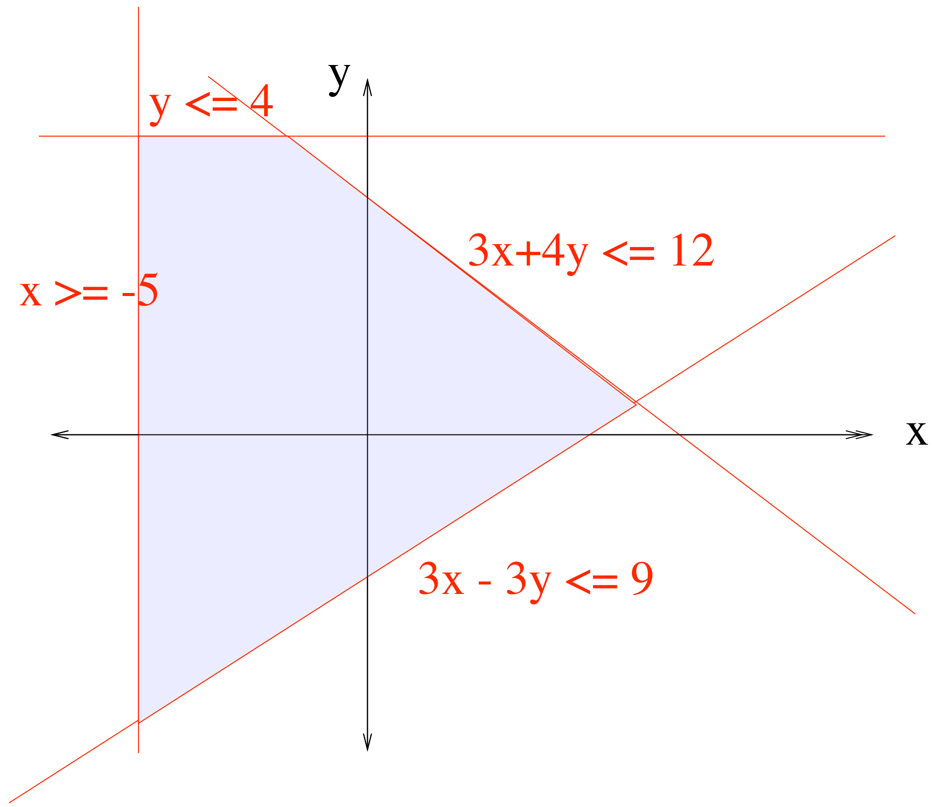
Inequality: half-plane (2D), half-space(>2D)



Region described by inequality is convex
(if two points are in region, all points in between them are in region)

Intuition about systems of linear inequalities:

Conjunction of inequalties = intersection of half-spaces
=> some convex region



Region described by inequalities is a convex polyhedron
(if two points are in region, all points in between them are in region)

Let us formulate correctness of loop permutation as ILP problem.

Intuition: If all iterations of a loop nest are independent, then permutation is certainly legal.

This is stronger than we need, but it is a good starting point.

What does independent mean?

Let us look at dependences.

Dependences:
- data
  - flow
  - anti
  - output
- control

Flow dependence: S1 -> S2

(i) S1 executes before S2 in program order

(ii) S1 writes into a location that is read by S2

Anti-dependence: S1 -> S2

(i) S1 executes before S2

(ii) S1 reads from a location that is overwritten later by S2

Output dependence: S1 -> S2

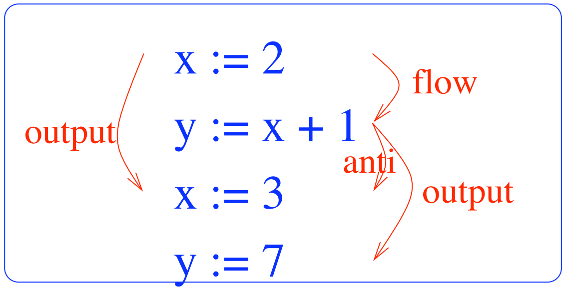
(i) S1 executes before S2

(ii) S1 and S2 write to the same location

Input dependence: S1 -> S2

(i) S1 executes before S2

(ii) S1 and S2 both read from the same location



Input dependence is not usually important for most applications.

Conservative Approximation:

- Real programs: imprecise information => need for safe approximation

  'When you are not sure whether a dependence exists, you must assume it does.'

Example:

```
procedure f (X,i,j)
 begin
   X(i) = 10;
   X(j) = 5;
  end
```

Question: Is there an output dependence from the first assignment to the second?

Answer: If (i = j), there is a dependence; otherwise, not.

=> Unless we know from interprocedural analysis that the parameters i and j are always distinct, we must play it safe and insert the dependence.

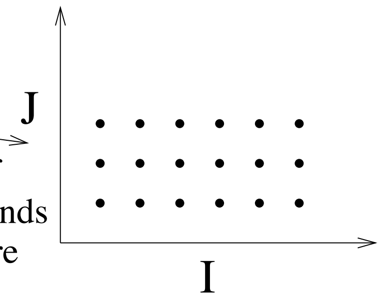
Key notion: Aliasing : two program names may refer to the same location (like X(i) and X(j))

May-dependence vs must-dependence: More precise analysis may eliminate may-dependences

Loop level Analysis: granularity is a loop iteration

```
DO I = 1, 100
  DO J = 1, 100
    S
```

each (I,J) value of
loop indices corresponds
to one point in picture



Dynamic instance of a statement:

Execution of a statement for given loop index values

Dependence between iterations:

Iteration (I1,J1) is said to be dependent on iteration (I2,J2) if
a dynamic instance (I1,J1) of a statement in loop body
is dependent on a dynamic instance (I2,J2) of a statement
in the loop body.

How do we compute dependences between iterations of a loop nest?

Dependences in loops

```
    FOR 10 I = 1, N
      X(f(I)) = ...
10            = ...X(g(I))..
```

- Conditions for flow dependence from iteration I_w to I_r:
  - $1 \leq I_w \leq I_r \leq N$ *(write before read)*
  - $f(I_w) = g(I_r)$ *(same array location)*
- Conditions for anti-dependence from iteration I_g to I_o:
  - $1 \leq I_g < I_o \leq N$ *(read before write)*
  - $f(I_o) = g(I_g)$ *(same array location)*
- Conditions for output dependence from iteration I_{w1} to I_{w2}:
  - $1 \leq I_{w1} < I_{w2} \leq N$ *(write in program order)*
  - $f(I_{w1}) = f(I_{w2})$ *(same array location)*

Dependences in nested loops

```
    FOR 10 I = 1, 100
     FOR 10 J = 1, 200
      X(f(I,J),g(I,J)) = ...
10               = ...X(h(I,J),k(I,J))..
```

Conditions for flow dependence from iteration (I_w, J_w) to (I_r, J_r):
Recall: $\preceq$ is the lexicographic order on iterations of nested loops.

$$
\begin{aligned}
1 &\leq I_w \leq 100 \\
1 &\leq J_w \leq 200 \\
1 &\leq I_r \leq 100 \\
1 &\leq J_r \leq 200 \\
(I_1, J_1) &\preceq (I_2, J_2) \\
f(I_1, J_1) &= h(I_2, J_2) \\
g(I_1, J_1) &= k(I_2, J_2)
\end{aligned}
$$

Anti and output dependences can be defined analogously.

Array subscripts are affine functions of loop variables

=>

dependence testing can be formulated as a set of ILP problems

ILP Formulation

```
FOR I = 1, 100
  X(2I) = .... X(2I+1)...
```

Is there a flow dependence between different iterations?

$$\begin{aligned} 1 &\leq Iw < Ir \leq 100 \\ 2Iw &= 2Ir + 1 \end{aligned}$$

which can be written as

$$\begin{aligned} 1 &\leq Iw \\ Iw &\leq Ir - 1 \\ Ir &\leq 100 \\ 2Iw &\leq 2Ir + 1 \\ 2Ir + 1 &\leq 2Iw \end{aligned}$$

The system

$$\begin{aligned} 1 &\leq Iw \\ Iw &\leq Ir - 1 \\ Ir &\leq 100 \\ 2Iw &\leq 2Ir + 1 \\ 2Ir + 1 &\leq 2Iw \end{aligned}$$

can be expressed in the form $Ax \leq b$ as follows

$$\begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 2 & -2 \\ -2 & 2 \end{pmatrix} \begin{bmatrix} Iw \\ Ir \end{bmatrix} \leq \begin{bmatrix} -1 \\ -1 \\ 100 \\ 1 \\ -1 \end{bmatrix}$$

ILP Formulation for Nested Loops

```
FOR I = 1, 100
 FOR J = 1, 100
   X(I,J) = ..X(I-1,J+1)...
```

Is there a flow dependence between different iterations?

$$
\begin{aligned}
1 &\leq Iw \leq 100 \\
1 &\leq Ir \leq 100 \\
1 &\leq Jw \leq 100 \\
1 &\leq Jr \leq 100 \\
(Iw, Jw) &\prec (Ir, Jr)(lexicographic\ order) \\
Ir - 1 &= Iw \\
Jr + 1 &= Jw
\end{aligned}
$$

Convert lexicographic order $\prec$ into integer equalities/inequalities.

$(Iw, Jw) \prec (Ir, Jr)$ is equivalent to

$Iw < Ir$ OR $((Iw = Ir)\ AND\ (Jw < Jr))$

We end up with two systems of inequalities:

$$
\begin{array}{l}
1 \le Iw \le 100 \\
1 \le Ir \le 100 \\
1 \le Jw \le 100 \\
1 \le Jr \le 100 \\
Iw < Ir \\
Ir - 1 = Iw \\
Jr + 1 = Jw
\end{array}
\qquad OR \qquad
\begin{array}{l}
1 \le Iw \le 100 \\
1 \le Ir \le 100 \\
1 \le Jw \le 100 \\
1 \le Jr \le 100 \\
Iw = Ir \\
Jw < Jr \\
Ir - 1 = Iw \\
Jr + 1 = Jw
\end{array}
$$

Dependence exists if either system has a solution.

What about affine loop bounds?

```
FOR I = 1, 100
  FOR J = 1, I
      X(I,J) = ..X(I-1,J+1)...
```

$$
\begin{aligned}
1 &\leq Iw \leq 100 \\
1 &\leq Ir \leq 100 \\
1 &\leq Jw \leq Iw \\
1 &\leq Jr \leq Ir \\
(Iw, Jw) &\prec (Ir, Jr)(lexicographicorder) \\
Ir - 1 &= Iw \\
Jr + 1 &= Jw
\end{aligned}
$$

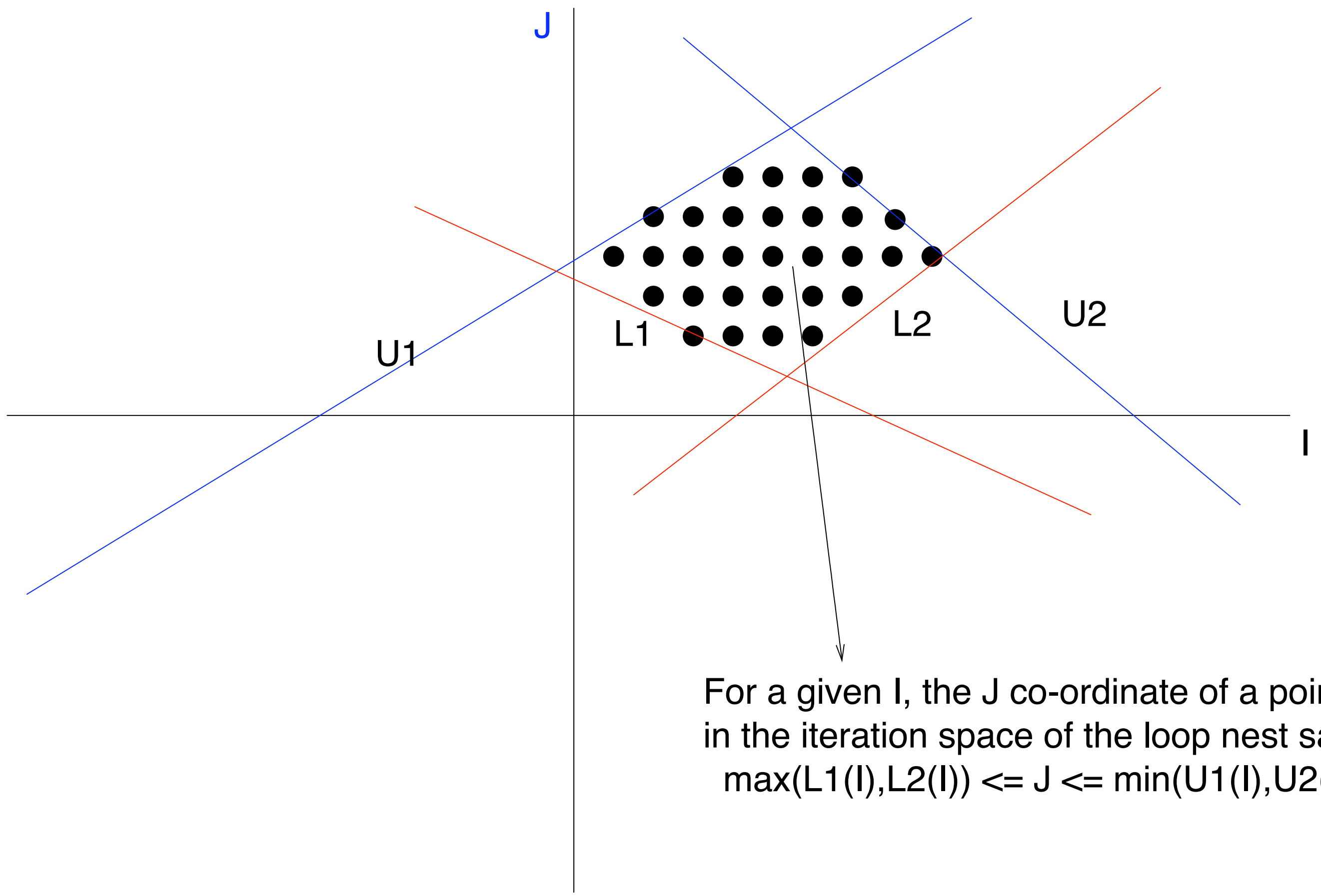
We can actually handle fairly complicated bounds involving min's and max's.

```
FOR I = 1, 100
  FOR J = max(F1(I),F2(I)) ,  min(G1(I),G2(I))
     X(I,J) = ..X(I-1,J+1)...
```

$$\begin{aligned} & \cdots \\ F1(Ir) &\leq Jr \\ F2(Ir) &\leq Jr \\ Jr &\leq G1(Ir) \\ Jr &\leq G2(Ir) \\ & \cdots \end{aligned}$$

Caveat: $F1, F2$ etc. must be affine functions.

Min's and max's in loop bounds mayseem weird, but actually they describe general polyhedral iteration spaces!



For a given I, the J co-ordinate of a point
in the iteration space of the loop nest satisfies
max(L1(I),L2(I)) <= J <= min(U1(I),U2(I))

More important case in practice: variables in upper/lower bounds

```
FOR I = 1, N
  FOR J = 1 , N-1
    ....
```

Solution: Treat N as though it was an unknown in system

$$
\begin{aligned}
1 &\leq Iw \leq N \\
1 &\leq Jw \leq N-1 \\
\ldots.
\end{aligned}
$$

This is equivalent to seeing if there is a solution for any value of N.

Note: if we have more information about the range of N, we can easily add it as additional inequalities.

Summary

Problem of determining if a dependence exists between two iterations of a perfectly nested loop can be framed as ILP problem of the form

Is there an integer solution to system $Ax \leq b$?

How do we solve this decision problem?

Is there an integer solution to system $Ax \leq b$?

Oldest solution technique: Fourier-Motzkin elimination

Intuition: "Gaussian elimination for inequalties"

More modern techniques exist, but all known solutions require time exponential in the number of inequalities

=>

Anything you can do to reduce the number of inequalities is good.

=>

Equalities should not be converted blindly into inequalities but handled separately.

Presentation sequence:

- one equation, several variables

  $2x + 3y = 5$

- several equations, several variables

  $$\begin{aligned} 2x + 3y + 5z &= 5 \\ 3x + 4y &= 3 \end{aligned}$$

- equations & inequalities

  $$\begin{aligned} &2x + 3y = 5 \\ &x \le 5 \\ &y \le -9 \end{aligned}$$

Diophatine equations:
use integer Gaussian
elimination

Solve equalities first
then use Fourier-Motzkin
elimination

One equation, many variables:

Thm: The linear Diophatine equation a1 x1 + a2 x2 + ...+ an xn = c has integer solutions iff gcd(a1,a2,...,an) divides c.

Examples:

(1) 2x = 3 No solutions

(2) 2x = 6 One solution: x = 3

(3) 2x + y = 3

GCD(2,1) = 1 which divides 3.

Solutions: x = t, y = (3 - 2t)

(4) 2x + 3y = 3

GCD(2,3) = 1 which divides 3.

Let z = x + floor(3/2) y = x + y

Rewrite equation as 2z + y = 3

Solutions: z = t, y = (3 - 2t) => x = (3t - 3), y = (3 - 2t)

Intuition: Think of underdetermined systems of eqns over reals.

Caution: Integer constraint => Diophantine system may have no solns

Thm: The linear Diophatine equation $a1\ x1 + a2\ x2 + ... + an\ xn = c$
has integer solutions iff gcd(a1,a2,...,an) divides c.

Proof: WLOG, assume that all coefficients a1,a2,...an are positive.
We prove only the IF case by induction, the proof in the other direction is trivial.
Induction is on min(smallest coefficient, number of variables).

Base case:

If (# of variables = 1) , then equation is a1 x1 = c which has integer solutions if a1 divides c.

If (smallest coefficient = 1), then gcd(a1,a2,...,an) = 1 which divides c.

Wlog, assume that a1 = 1, and observe that the equation has solutions of the form (c - a2 t2 - a3 t3 -....-an tn, t2, t3, ...tn).

Inductive case:

Suppose smallest coefficient is a1, and let $t = x1 + \text{floor}(a2/a1)\ x2 + ... + \text{floor}(an/a1)\ xn$

In terms of this variable, the equation can be rewritten as

$(a1)\ t + (a2 \bmod a1)\ x2 + ... + (an \bmod a1)\ xn = c$ (1)

where we assume that all terms with zero coefficient have been deleted.

Observe that (1) has integer solutions iff original equation does too.

Now gcd(a,b) = gcd(a mod b, b) => gcd(a1,a2,...,an) = gcd(a1, (a2 mod a1),..,(an mod a1))
=> gcd(a1, (a2 mod a1),..,(an mod a1)) divides c.

If a1 is the smallest co-efficient in (1), we are left with 1 variable base case.

Otherwise, the size of the smallest co-efficient has decreased, so we have made progress in the induction.

Summary:

$$\text{Eqn:} \quad a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = c$$

- Does this have integer solutions?

= Does gcd(a1,a2,...,an) divide c ?

It is useful to consider solution process in matrix-theoretic terms.

We can write single equation as

$$(3\ 5\ 8)(x\ y\ z)^T \quad = \ 6$$

It is hard to read off solution from this, but for special matrices, it is easy.

$$(2\ \ 0)(a\ \ b)^T \quad = 8$$

Solution is a = 4, b = t

looks lower triangular, right?

Key concept: column echelon form -
"lower triangular form for underdetermined systems"

For a matrix with a single row, column echelon form is
(x 0 0 0...0)

$3x + 5y + 8z = 6$

Substitution: $t = x + y + 2z$
New equation:

$3t + 2y + 2z = 6$

Substitution: $u = y+z+t$
New equation:

$2u + t = 6$

Solution:
$u = p1$
$t = (6-2p1)$

Backsubstitution:
$y = p2$
$t = (6-2p1)$
$z = (3p1-p2-6)$

Backsubstitution:
$x = (18-8p1+p2)$
$y = p2$
$z = (3p1-p2-6)$

$(3\ 5\ 8)$

$$(3\ 5\ 8)\begin{pmatrix} 1 & -1 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{U1}$$

$= (3\ 2\ 2)$

$$(3\ 2\ 2)\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \text{U2}$$

$= (1\ 2\ 0)$

$$(1\ 2\ 0)\begin{pmatrix} 1 & -2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{U3}$$

$= (1\ 0\ 0)$

Solution: $(6\ a\ b)^T$

U1*U2*U3

$$\text{Product of matrices} = \begin{pmatrix} 2 & -5 & -1 \\ -1 & 3 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

Solution to original system:
$U1*U2*U3*(6\ a\ b)^T$

$$\begin{pmatrix} 12-5a-b \\ -6+3a-b \\ b \end{pmatrix}$$

Systems of Diophatine Equations:

Key idea: use integer Gaussian elimination

Example:

$$\begin{aligned} 2x + 3y + 4z &= 5 \\ x - y + 2z &= 5 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 2 & 3 & 4 \\ 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

It is not easy to determine if this Diophatine system has solutions.

Easy special case: lower triangular matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 5 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} x &= 5 \\ y &= 3 \\ z &= \text{arbitrary integer} \end{aligned}$$

Question: Can we convert general integer matrix into equivalent lower triangular system?

INTEGER GAUSSIAN ELIMINATION

Integer gaussian Elimination

- Use row/column operations to get matrix into triangular form
- For us, column operations are more important because we usually have more unknowns than equations

Overall strategy: Given Ax = b

Find matrices U1, U2,...Uk such that

A*U1*U2*...*Uk is lower triangular (say L)

Solve Lx' = b (easy)

Compute x = (U1*U2*...*Uk)*x

Proof:

(A*U1*U2...*Uk)x' = b

=> A(U1*U2*...*Uk)x' = b

=> x = (U1*U2...*Uk)x'

Caution: Not all column operations preserve integer solutions.

$$\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$ Solution: x = -8, y = 7

$$\downarrow \begin{bmatrix} 1 & -3 \\ 0 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 \\ 6 & -4 \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$ which has no integer solutions!

Intuition: With some column operations, recovering solution of original system requires solving lower triangular system using rationals.

Question: Can we stay purely in the integer domain?

One solution: Use only unimodular column operations

Unimodular Column Operations:

(a) Interchange two columns

$$\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \xrightarrow[\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}]{} \begin{bmatrix} 3 & 2 \\ 7 & 6 \end{bmatrix}$$

Check

Let x,y satisfy first eqn.

Let x',y' satisfy second eqn.

$x' = y\ , \quad y' = x$

(b) Negate a column

$$\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \xrightarrow[\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}]{} \begin{bmatrix} 2 & -3 \\ 6 & -7 \end{bmatrix}$$

Check

$x' = x, \quad y' = -y$

(c) Add an integer multiple of one column to another

$$\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \xrightarrow[\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}]{} \begin{bmatrix} 2 & 1 \\ 6 & 1 \end{bmatrix}$$

$n = -1$

Check

$x = x' + n\, y'$

$y = y'$

Example:

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 3 & 0 \\ 1 & -1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 1 & 0 \\ 1 & -2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 5 & -2 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ -2 & 5 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 5 & 0 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \Rightarrow \begin{array}{l} x' = 5 \\ y' = 3 \\ z' = t \end{array} \Rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1 & 3 & -2 \\ 1 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ t \end{bmatrix} = \begin{bmatrix} 4-2t \\ -1 \\ t \end{bmatrix}$$

Facts:

1. The three unimodular column operations
   - interchanging two columns
   - negating a column
   - adding an integer multiple of one column to another

   on the matrix A of the system A x = b
   preserve integer solutions, as do sequences of these operations.

2. Unimodular column operations can be used to reduce a matrix A into lower triangular form.

3. A unimodular matrix has integer entries and a determinant of +1 or -1.

4. The product of two unimodular matrices is also unimodular.

Algorithm: Given a system of Diophantine equations $Ax = b$

1. Use unimodular column operations to reduce matrix A to lower triangular form L.
2. If $Lx' = b$ has integer solutions, so does the original system.
3. If explicit form of solutions is desired, let U be the product of unimodular matrices corresponding to the column operations.

   $x = U x'$ where x' is the solution of the system $Lx' = b$

Detail: Instead of lower triangular matrix, you should to compute 'column echelon form' of matrix.

Column echelon form: Let rj be the row containing the first non-zero in column j.
(i) $r(j+1) > rj$ if column j is not entirely zero.
(ii) column (j+1) is zero if column j is.

$$\begin{bmatrix} x & 0 & 0 \\ x & 0 & 0 \\ x & x & x \end{bmatrix}$$

is lower triangular but not column echelon.
Point: writing down the solution for this system requires additional work with the last equation (1 equation, 2 variables). This work is precisely what is required to produce the column echelon form.

Note: Even in regular Gaussian elimination, we want column echelon form rather than lower triangular form when we have under-determined systems.