Transformations of Imperfectly Nested loops

Induprakas Kodukula
Keshav Pingali

This research was supported by an NSF Presidential Young Investigator award CCR-8958543, NSF grant CCR-9008526, ONR grant N00014-93-1-0103, and a grant from Hewlett-Packard Corporation.

Organization

- Linear framework
- Transformations
- Code generation and Completion
- Related Work and Conclusions

Imperfectly Nested Loops

Loop is perfectly nested if

- body is completely surrounded by all the loops

Loop is imperfectly nested if

- it is not perfectly nested

```
for I = 1 to N do
  S1: A[i] = sqrt (A[I])
  for J = I+1 to N do
    S2: A[J] = A[J] / A[I]
  endfor
endfor
```

Existing Framework - perfectly nested loops

What is nice about it?

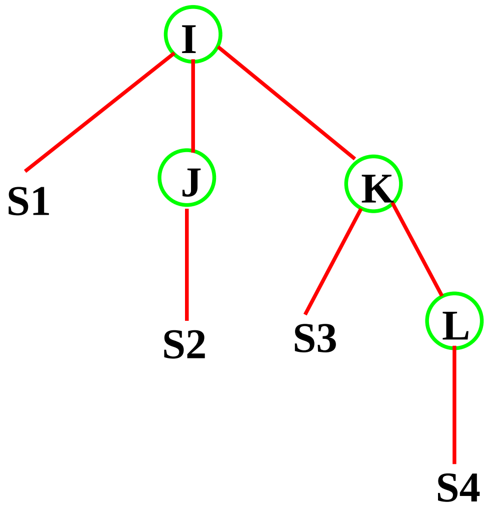
- Execution mapped to integer space modeling program execution order
- Transformation as matrices
- Composition
- Simple legality test
- Automatic code generation

Banerjee, Li and Pingali, Padua, Wolf and Lam,....

Imperfect Loop as AST

- Internal nodes $\equiv$ loops
- Edges $\equiv$ parent-child relationship
- Leaves $\equiv$ atomic statements

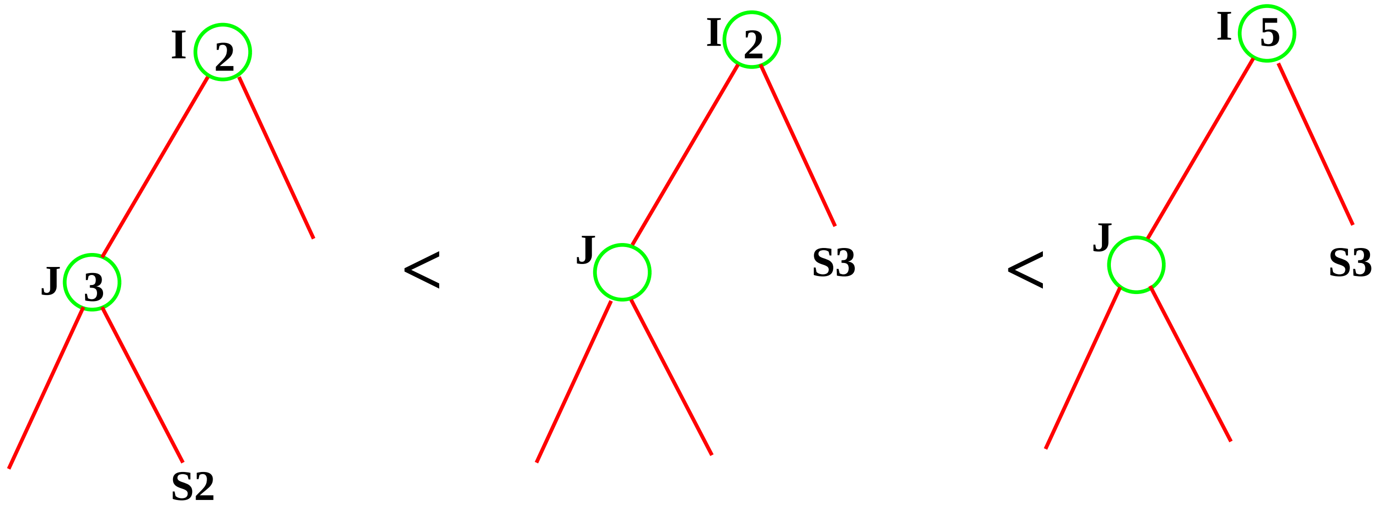
```
for I
    S1
    for J
        S2
    for K
        S3
        for L
            S4
```



Desired - Encode statement execution

Want a vector encoding which:

- satisfies uniqueness
- preserves execution order as lexicographic order



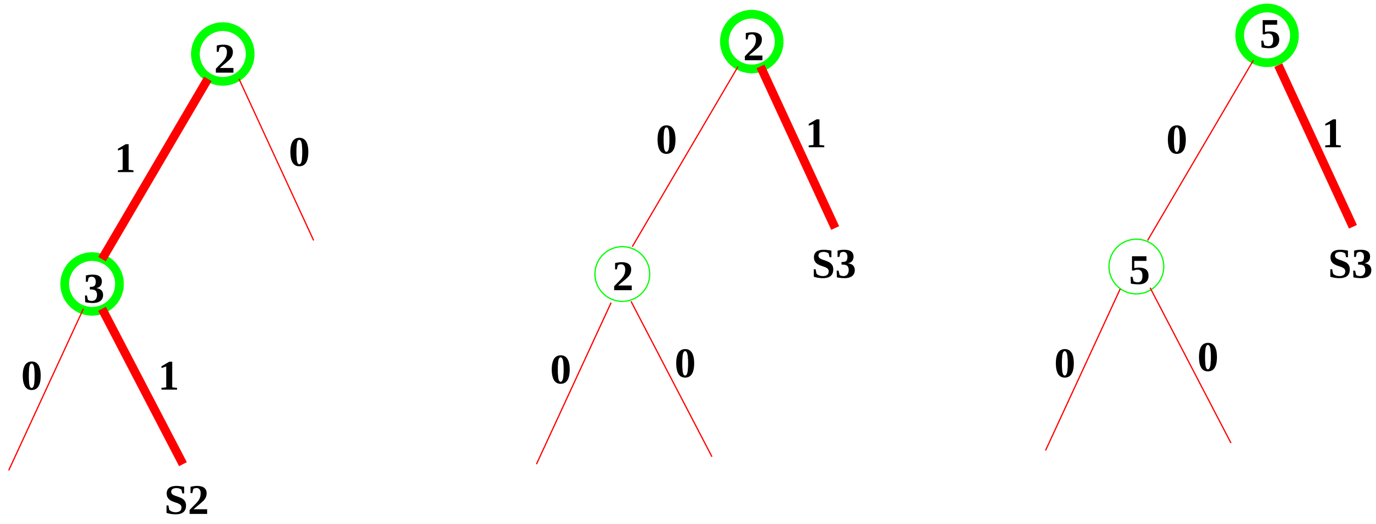
Rules to label the AST

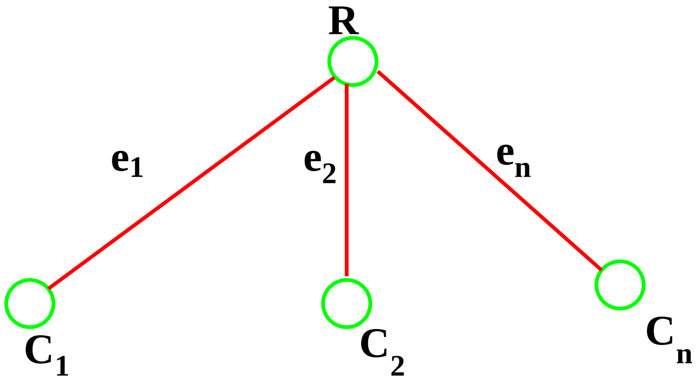
$\mathfrak{M}_e =$

- 1 if e is marked
- 0 otherwise

$\mathfrak{M}_n =$

- loopvar value if n is marked
- loopvar value of first marked ancestor otherwise





for leaf nodes

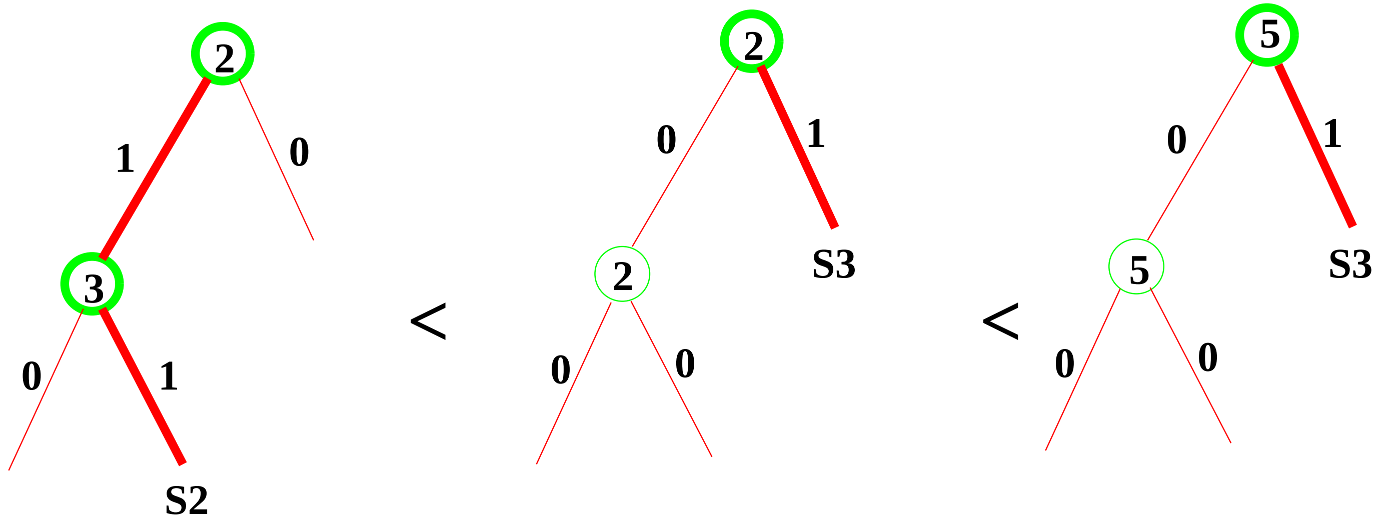
- $\mathfrak{L}(R) = \epsilon$

For internal nodes

- $\mathfrak{L}(R) =$
  $\mathfrak{M}_R // \mathfrak{M}(e_n) // \ldots // \mathfrak{M}(e_2) // \mathfrak{M}(e_1) // \mathfrak{L}(C_n) // \ldots // \mathfrak{L}(C_2) // \mathfrak{L}(C_1)$

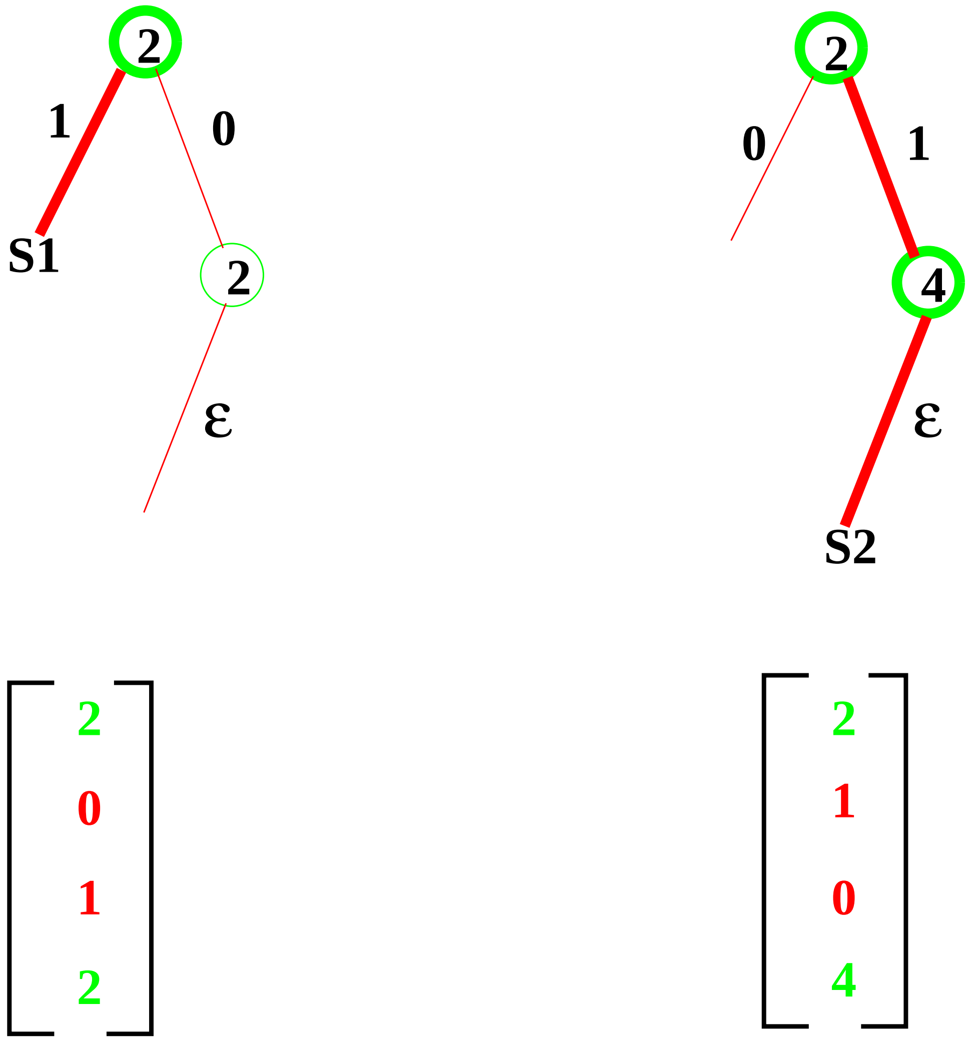
Labeled AST $\equiv \mathfrak{L}(root) \equiv$ Instance Vector

Putting it all together



$$\begin{bmatrix} 2 \\ 0 \\ 1 \\ 3 \\ 1 \\ 0 \end{bmatrix} < \begin{bmatrix} 2 \\ 1 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} < \begin{bmatrix} 5 \\ 1 \\ 0 \\ 5 \\ 0 \\ 0 \end{bmatrix}$$

Optimizing single edges

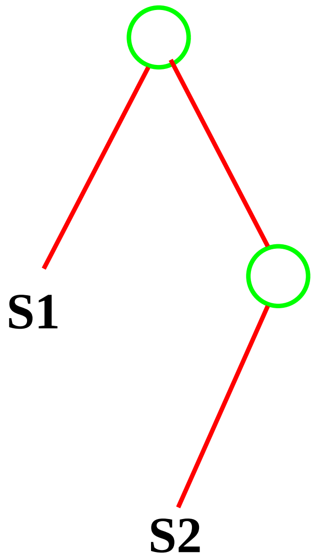


Properties of Instance Vectors

- Program execution order $\equiv$ lexicographic order
- Flow dependence $= IV_{read} - IV_{write}$, etc
- Distance/direction vectors are lexicographically positive

Computing dependences

```
for I = 1..N
    S1: A[I]= sqrt (A[I])
    for J = I+1..N
        S2: A[J] = A[J]/A[I]
```



Flow S1→S2 on A[I]: $\begin{bmatrix} I_r \\ 1 \\ 0 \\ J_r \end{bmatrix} (read) - \begin{bmatrix} I_w \\ 0 \\ 1 \\ I_w \end{bmatrix} (write)$

```
for I = 1 to N do
  S1:  A[I]  = sqrt (A[I]);
  for J = I+1 to N do
    S2: A[J] = A[J] /  A[I] ;
```

$$
\begin{aligned}
1 \leq I_r \leq N, 1 \leq I_w \leq N, I_r < J_r \\
I_w \leq I_r \\
I_r = I_w \\
\Delta 1 = I_r - I_w, \Delta 2 = J_r - I_w
\end{aligned} \tag{1}
$$

$$
\begin{bmatrix} I_r \\ 1 \\ 0 \\ J_r \end{bmatrix} - \begin{bmatrix} I_w \\ 0 \\ 1 \\ I_w \end{bmatrix} = \begin{bmatrix} I_r - I_w \\ 1 \\ -1 \\ J_r - I_w \end{bmatrix} = \begin{bmatrix} \Delta 1 \\ 1 \\ -1 \\ \Delta 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ + \end{bmatrix}
$$

- Model
- Transformations
- Code generation and Completion
- Related Work and Conclusions

Transformations

- Following transformations:
  - reversal, scaling, skewing and permutation
  - Distribution and Jamming
  - Statement alignment and reordering
- Individual transformations $\equiv$ Matrices
- Legality: $T * d > 0$ (similar to perfectly nested loop case)

Permutation and reordering

```
for I = 1 to N do
    S1: A[I] = sqrt (A[I]);
    for J = I+1 to N do
        S2: A[J] = A[J] / A[I];
```

$$\Downarrow$$

```
for J = 1 to N do
    for I = 1 to J-1 do
        S2: A[J] = A[J] / A[I];
    S1: A[J] = sqrt (A[J]);
```

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

Legality

T is legal w.r.t d if $T * d > 0$ (lexicographically positive)

T must be legal w.r.t all dependences in the initial code

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \\ + & 0 & 1 \end{bmatrix} = \begin{bmatrix} + & 0 & 1 \\ -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- Can prove that all six versions of Cholesky are equivalent to each other

```
for I = 1 to N do
  S1: = .... A[I-4,I+2]
  for J = I to N do
    S2:  A[I,J]  = ...
  endfor
endfor
```

Flow dependence S2→S1:

$$\begin{bmatrix} I_r \\ 0 \\ 1 \\ I_r \end{bmatrix} - \begin{bmatrix} I_w \\ 1 \\ 0 \\ J_r \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \\ -2 \end{bmatrix}$$

Want parallel outer loop, i.e.

first entry of transformed dependence vector equal to 0

Statement Alignment

$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 \\ -1 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ -2 \end{bmatrix}$$

Corresponds to statement alignment

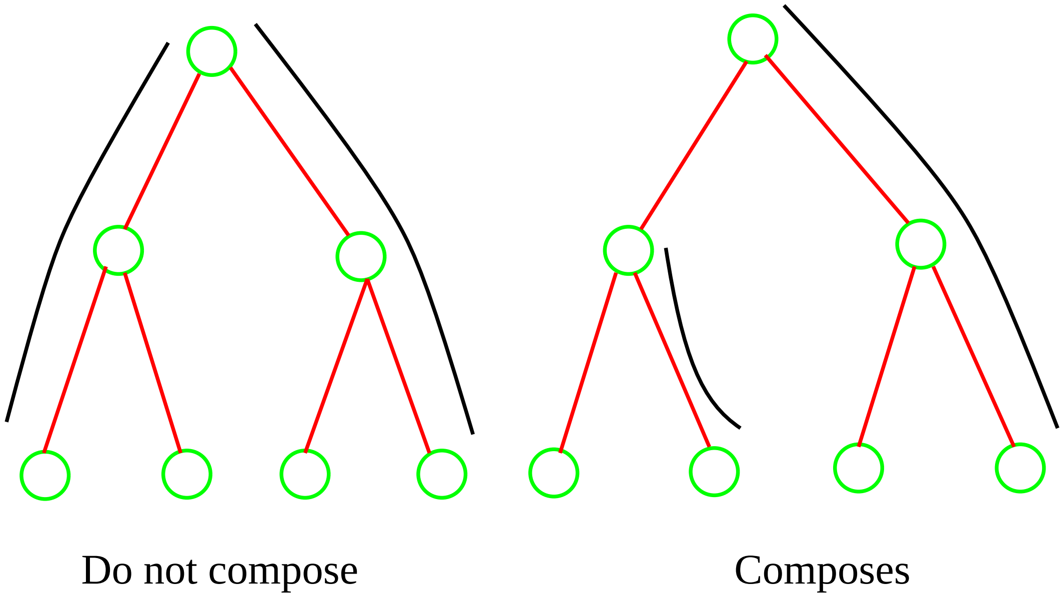
Loop Skewing

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 4 \\ -1 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ -2 \end{bmatrix}$$

Corresponds to skewing the outer loop by the inner loop

Composition of transformations

- Multiple linear transformations on same path of AST compose
- Statement reordering/distribution/jamming compose with linear transformations.
- Linear transformations on disjoint paths compose
- Linear transformations on intersecting paths **do not** compose.



- Model
- Transformations
- Code Generation and Completion
- Related Work and Conclusions

Code generation

Input:

- Input program and matrix of transformation

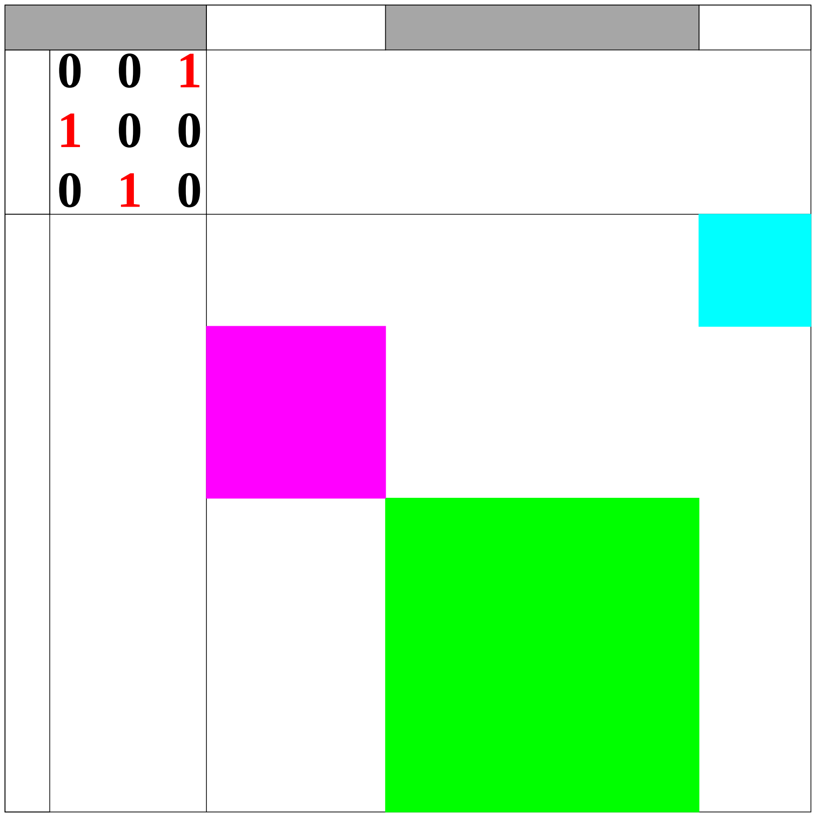
Output:

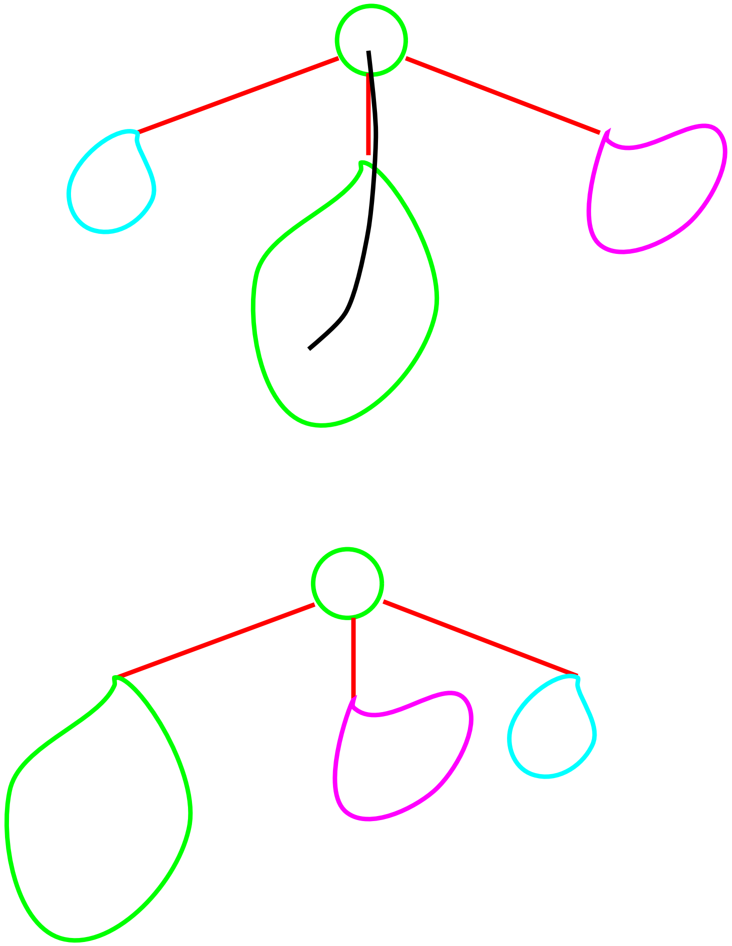
- Transformed program

How?

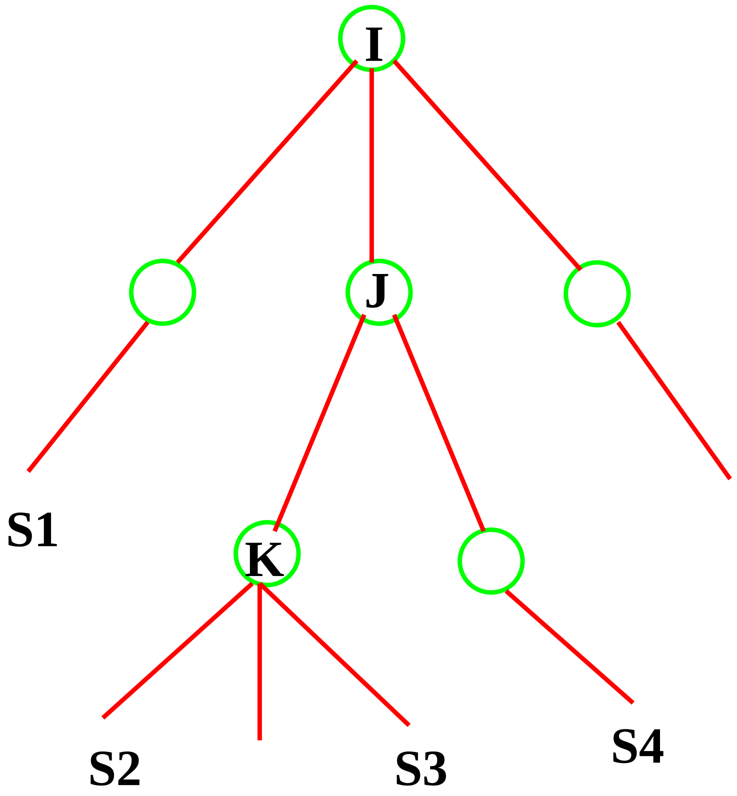
- Build transformed syntax tree
- For each statement, obtain bounds for surrounding loops
- Merge bounds for loops containing multiple statements and insert guards as necessary

Shape of a transformation





Discovering AST structure is a recursive problem



Definition:

- A dependence between S and D is *contained* in the deepest common ancestor of S and D

Completion Procedure

Input:

- Legal partial transformation & unsatisfied dependences

Output:

- Legal complete transformation

How?

- Generate a block at a time
- Ensure permutation for immediate children
- Must satisfy contained dependences
- Must not violate all other dependences

Linear algebra formulation

Given two sets of dependence vectors $\mathfrak{D}_1$ and $\mathfrak{D}_2$

Find a vector v such that

$$\begin{aligned} \overrightarrow{v} * \overrightarrow{d_1}^T &\geq 0, d_1 \in \mathfrak{D}_1 \\ \overrightarrow{v} * \overrightarrow{d_2}^T &> 0, d_2 \in \mathfrak{D}_2 \end{aligned} \tag{2}$$

First step - Obtain dependences

```
for I = 1 to N do
  A[I] = sqrt (A[I]);
  for J = I+1 to N do
    A[J] = A[J] / A[I];
  endfor
endfor
```

Dependence Matrix is $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \\ + & 0 & 1 \end{bmatrix}$

Interchange of I and J loops

Suppose want J loop outermost:

- First row of the transformation is $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$

Next block can be $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ or $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \\ + & 0 & 1 \end{bmatrix} = \begin{bmatrix} + & 0 & 1 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

Block row must be $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

Complete transformation is $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

Code generated after transformation:

```
for J = 1 to N do
  for I = 1 to J-1 do
    A[J] = A[J] / A[I];
  endfor
  A[J] = sqrt (A[J]);
endfor
```

Related Work

- Started with Wolfe's work - direct interchange
- Statement Alignment (Callahan, Ayguade)
- Schedules (Feautrier) and Mappings (Kelly, Pugh)

	Mappings	Our approach
Expressiveness	more	less
Dependence	exact, transitive closure	distance, direction
Code generation	Need to rediscover structure	somewhat simpler
Completion procedure	Search	Similar to perfectly nested case

Conclusions and Future Work

- Extend Completion Procedure to include distribution and jamming
- Remove disjoint paths restriction from completion procedure

Have a Polaris implementation which

- Does dependence analysis
- Does code generation given a user input transformation
- Currently implementing completion procedure