iscc Tutorial

Sven Verdoolaege

INRIA, France and KU Leuven
Sven.Verdoolaege@inria.fr

December 19, 2014
Outline

1. Introduction

2. Basic Concepts and Operations
   - Sets and Statement Instances
   - Maps and AST Generation
   - Access Relations and Polyhedral Model
   - Dataflow Analysis
   - Transitive Closures
   - Basic Counting
   - Computing Bounds
   - Weighted Counting

3. Simple Applications
   - Pointer Conversion
   - Dynamic Memory Requirement Estimation
   - Reuse Distance Computation
Outline

1 Introduction

2 Basic Concepts and Operations
   - Sets and Statement Instances
   - Maps and AST Generation
   - Access Relations and Polyhedral Model
   - Dataflow Analysis
   - Transitive Closures
   - Basic Counting
   - Computing Bounds
   - Weighted Counting

3 Simple Applications
   - Pointer Conversion
   - Dynamic Memory Requirement Estimation
   - Reuse Distance Computation
Introduction

What is iscc?

⇒ interactive interface to the barvinok counting library
⇒ also provides interface to the pet polyhedral model extractor and to some operations from the isl integer set library, including AST generation
⇒ inspired by Omega Calculator from the Omega Project
Introduction

- What is iscc?
  - interactive interface to the barvinok counting library
  - also provides interface to the pet polyhedral model extractor and to some operations from the isl integer set library, including AST generation
  - inspired by Omega Calculator from the Omega Project

- Where to get iscc?
  - currently distributed as part of barvinok package
  - available from http://barvinok.gforge.inria.fr/
Introduction

• What is iscc?
  ⇒ interactive interface to the barvinok counting library
  ⇒ also provides interface to the pet polyhedral model extractor and to some operations from the isl integer set library, including AST generation
  ⇒ inspired by Omega Calculator from the Omega Project

• Where to get iscc?
  ⇒ currently distributed as part of barvinok package
  ⇒ available from http://barvinok.gforge.inria.fr/

• How to run iscc?
  ⇒ compile and install barvinok following the instructions in README
  ⇒ run iscc
  Note: iscc currently does not use readline, so you may want to use a readline front-end: rlwrap iscc
Introduction

- **What is iscc?**
  - interactive interface to the barvinok counting library
  - also provides interface to the pet polyhedral model extractor and to some operations from the isl integer set library, including AST generation
  - inspired by Omega Calculator from the Omega Project

- **Where to get iscc?**
  - currently distributed as part of barvinok package
  - available from http://barvinok.gforge.inria.fr/

- **How to run iscc?**
  - compile and install barvinok following the instructions in README
  - run iscc
    Note: iscc currently does not use readline, so you may want to use a readline front-end: rlwrap iscc

Examples from polyhedral model for program analysis and transformation
Interaction with Libraries and Tools

- **LLVM**
- **imath**
- **GMP**
- **isl**
- **NTL**
- **PolyLib**
- **clang**
- **pet**
- **PPCG**
- **isa**
- **iscc**

**isl**: manipulates parametric affine sets and relations

**barvinok**: counts elements in parametric affine sets and relations

**pet**: extracts polyhedral model from clang AST

**PPCG**: Polyhedral Parallel Code Generator

**iscc**: interactive calculator

**isa**: prototype tool set including derivation of process networks and equivalence checker

**Licenses**: BSD/MIT, LGPL, GPL
Overview of isl

isl is a thread-safe C library for manipulating integer sets and relations

- bounded by *affine constraints*
- involving *symbolic constants* and
- *existentially quantified variables*

and *quasi-affine* and *quasi-polynomial functions* on such domains

Supported operations by core library include

- *intersection*
- *union*
- *set difference*
- *integer projection*
- *coalescing*
- *closed convex hull*
- *sampling*, *scanning*
- *integer affine hull*
- *lexicographic optimization*
- *transitive closure* (approx.)
- *parametric vertex enumeration*
- *bounds on quasipolynomials*

Polyhedral compilation library

- *schedule trees*
- *dataflow analysis*
- *scheduling*
- *AST generation*
Outline

1 Introduction

2 Basic Concepts and Operations
   - Sets and Statement Instances
   - Maps and AST Generation
   - Access Relations and Polyhedral Model
   - Dataflow Analysis
   - Transitive Closures
   - Basic Counting
   - Computing Bounds
   - Weighted Counting

3 Simple Applications
   - Pointer Conversion
   - Dynamic Memory Requirement Estimation
   - Reuse Distance Computation
Statement Instance Set

```c
for (i = 1; i <= 5; ++i)
    for (j = 1; j <= i; ++j)
        /* S */
```
Statement Instance Set

\[
\text{for} \ (i = 1; i \leq 5; ++i)
\]
\[
\text{for} \ (j = 1; j \leq i; ++j)
\]
\[
/* S */
\]
Statement Instance Set

\[\text{for } (i = 1; i <= 5; ++i)\]
\[\text{for } (j = 1; j <= i; ++j)\]
\[/* S */\]

\{ S[i,j] : 1 <= i <= 5 and 1 <= j <= i \}
Statement Instance Set

\begin{verbatim}
for (i = 1; i <= 5; ++i)
  for (j = 1; j <= i; ++j)
    /* S */
\end{verbatim}

(optional) name of space

\{ S[i,j] : 1 <= i <= 5 and 1 <= j <= i \}
Statement Instance Set

\[ \text{for } (i = 1; i <= 5; ++i) \]
\[ \text{for } (j = 1; j <= i; ++j) \]
\[ /* S */ \]

(optional) name of space

\{ S[i,j] : 1 <= i <= 5 and 1 <= j <= i \}

set variables
Statement Instance Set

\[ \text{for } (i = 1; i \leq 5; ++i) \]
\[ \text{for } (j = 1; j \leq i; ++j) \]
\[ /* S */ \]

(optional) name of space

\[ \{ S[i,j] : 1 \leq i \leq 5 \text{ and } 1 \leq j \leq i \} \]

set variables Presburger formula
Statement Instance Set

```
for (i = 1; i <= n; ++i)
    for (j = 1; j <= i; ++j)
        /* S */
```

(optional) name of space

\[ [n] \rightarrow \{ S[i,j] : 1 \leq i \leq n \text{ and } 1 \leq j \leq i \} \]
Statement Instance Set

```
for (i = 1; i <= n; ++i)
  for (j = 1; j <= i; ++j)
    /* S */
```

(optional) name of space

```
[n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i }
```

symbolic constants set variables Presburger formula
Set Variables and Symbolic Constants

- set variables
  - local to set
  - identified by position

- symbolic constants
  - global
  - identified by name
Set Variables and Symbolic Constants

- set variables
  - local to set
  - identified by position

- symbolic constants
  - global
  - identified by name

\[ [n] \rightarrow \{ [i,j] : 1 \leq i \leq n \text{ and } 1 \leq j \leq i \} \]

is equal to
\[ [n] \rightarrow \{ [a,b] : 1 \leq a \leq n \text{ and } 1 \leq b \leq a \} \]

but not equal to
\[ [n] \rightarrow \{ [j,i] : 1 \leq i \leq n \text{ and } 1 \leq j \leq i \} \]
or
\[ [m] \rightarrow \{ [i,j] : 1 \leq i \leq m \text{ and } 1 \leq j \leq i \} \]
AST Generation, Schedules and Maps

\textbf{for} (i = 1; i <= n; ++i)
  \textbf{for} (j = 1; j <= i; ++j)
  /* S */

codegen [n] -> \{ S[i,j] : 1 <= i <= n and 1 <= j <= i \};
⇒ generate AST that visits elements in lexicographic order
**AST Generation, Schedules and Maps**

```c
for (i = 1; i <= n; ++i)
    for (j = 1; j <= i; ++j)
        /* S */
```

```c
codegen [n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i };
⇒ generate AST that visits elements in lexicographic order
```

What if a different order is needed?

⇒ apply a **schedule**: maps instance set to multi-dimensional time

⇒ multi-dimensional time is ordered lexicographically

Example: interchange i and j

```c
{S[i,j] -> [t1,t2] : t1 = j and t2 = i}
```
For $i = 1; i \leq n; ++i$
   For $j = 1; j \leq i; ++j$
   /* S */

codegen $[n] \rightarrow \{ S[i,j] : 1 \leq i \leq n \text{ and } 1 \leq j \leq i \};$

⇒ generate AST that visits elements in lexicographic order

What if a different order is needed?
⇒ apply a schedule: maps instance set to multi-dimensional time
⇒ multi-dimensional time is ordered lexicographically

Example: interchange $i$ and $j$
\{S[i,j] \rightarrow [t1,t2] : t1 = j \text{ and } t2 = i\} \text{ or } \{S[i,j] \rightarrow [j,i]\}
AST Generation, Schedules and Maps

```c
for (i = 1; i <= n; ++i)
    for (j = 1; j <= i; ++j)
        /* S */
```

codegen [n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i };

⇒ generate AST that visits elements in lexicographic order

What if a different order is needed?
⇒ apply a schedule: maps instance set to multi-dimensional time
⇒ multi-dimensional time is ordered lexicographically

Example: interchange i and j
{S[i,j] -> [t1,t2] : t1 = j and t2 = i} or {S[i,j] -> [j,i]}
S := [n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i };
codegen ({S[i,j] -> [j,i]} * S);
AST Generation, Schedules and Maps

```c
for (i = 1; i <= n; ++i)
  for (j = 1; j <= i; ++j)
    /* S */

codegen [n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i };
⇒ generate AST that visits elements in lexicographic order

What if a different order is needed?
⇒ apply a schedule: maps instance set to multi-dimensional time
⇒ multi-dimensional time is ordered lexicographically

Example: interchange i and j
{S[i,j] -> [t1,t2] : t1 = j and t2 = i} or {S[i,j] -> [j,i]}
S := [n] -> { S[i,j] : 1 <= i <= n and 1 <= j <= i };
codegen ({S[i,j] -> [j,i]} * S);
```

intersect domain of map on the left with set on the right
Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names
AST Generation, Schedules and Maps

Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names

Examples:

\[
S := [n] \rightarrow \{ A[i] : 0 \leq i \leq n; B[i] : 0 \leq i \leq n \};
\]
\[
M := \{ A[i] \rightarrow [0,i]; B[i] \rightarrow [1,i] \};
\]
\[
\text{codegen (M * S)};
\]
AST Generation, Schedules and Maps

Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names

Examples:

\[
S := [n] \rightarrow \{ A[i] : 0 \leq i \leq n; B[i] : 0 \leq i \leq n \};
\]

\[
M := \{ A[i] \rightarrow [0,i]; B[i] \rightarrow [1,i] \};
\]

codegen (M * S);
AST Generation, Schedules and Maps

Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names

Examples:

\[
S := [n] \rightarrow \{ A[i] : 0 \leq i \leq n; B[i] : 0 \leq i \leq n \};
\]
\[
M := \{ A[i] \rightarrow [i, 0]; B[i] \rightarrow [i, 1] \};
\]
codegen (M * S);

all elements of A before any element of B
AST Generation, Schedules and Maps

Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names

Examples:

S := [n] -> { A[i] : 0 <= i <= n; B[i] : 0 <= i <= n };  
M := { A[i] -> [0,i]; B[i] -> [1,i] };  
codegen (M * S);

codegen3, codegen4
Generating AST for more than one space/statement

⇒ spaces should be named to distinguish them from each other
⇒ schedule is required because no ordering defined over spaces with different names

Examples:

S := [n] -> { A[i] : 0 <= i <= n; B[i] : 0 <= i <= n };  
M := { A[i] -> [0,i]; B[i] -> [1,i] };  
codegen (M * S);

deletion

each element of A after corresponding element of B

disjunction

S := [n] -> { A[i] : 0 <= i <= n; B[i] : 0 <= i <= n };  
M := { A[i] -> [i,1]; B[i] -> [i,0] };  
codegen (M * S);
Access Relations and Polyhedral Model

Simple program with temporary array $t$:

```c
for (i = 0; i < N; ++i)
S1:   t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:   b[i] = g(t[N-i-1]);
```

An access relation maps a statement instance to an array index.
For example, the access relation for the read in $S2$:

$[N] \rightarrow \{ S2[i] \rightarrow t[N-i-1] \}$
Access Relations and Polyhedral Model

Simple program with temporary array \( t \):

```c
for (i = 0; i < N; ++i)
S1: \( t[i] = f(a[i]); \)
for (i = 0; i < N; ++i)
S2: \( b[i] = g(t[N-i-1]); \)
```

An access relation maps a statement instance to an array index
For example, the access relation for the read in \( S2 \):

\[
[N] \rightarrow \{ S2[i] \rightarrow t[N-i-1] \}
\]

Polyhedral model of a program consists of

- statement instance set
- access relations (must writes, may writes, reads)
- initial schedule

\[
M := \text{parse_file("simple.c")};
D := M[0]; W := M[1]; R := M[3]; S := M[4];
\]
Lexicographic Optimization

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- What is the last iteration of the loop?

```c
S := [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
lexmax S;
```
Lexicographic Optimization

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- What is the last iteration of the loop?

```
S := [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
lexmax S;
```

lexicographically last element of set
Lexicographic Optimization

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- What is the last iteration of the loop?
  
  \( S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} \);
  \( \text{lexmax } S; \) lexicographically last element of set

- When is a given array element accessed last?
  
  \( A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \)
  \( \text{lexmax } (A^{-1}); \)
**Lexicographic Optimization**

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- What is the last iteration of the loop?
  
  \[ S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} ; \]
  
  \[ \text{lexmax } S ; \] lexicographically last element of set

- When is a given array element accessed last?
  
  \[ A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} ; \]
  
  \[ \text{lexmax } (A^{-1}) ; \] inverse map

`lex1, lex2`
Lexicographic Optimization

```plaintext
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- What is the last iteration of the loop?
  
  \[
  S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \};
  \]
  \[\text{lexmax } S;\]
  lexicographically last element of set

- When is a given array element accessed last?
  
  \[
  A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \};
  \]
  \[\text{lexmax } (A^{-1});\]
  inverse map
  lexicographically last image element
Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

Simple case: array written through a single access

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        F: a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
    W: Write(a[i]);
```

Access relations:

- \( A_1 := [N] \rightarrow \{ F[i,j] -> a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N \} \)
- \( A_2 := [N] \rightarrow \{ W[i] -> a[i] : 0 \leq i < N \} \)

Map to all writes:

\( R := A_2 . (A_1^{-1}) \)

Last write:

\( \text{lexmax } R \)

In general: impose lexicographical order on shared iterators
Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

Simple case: array written through a single access

\[
\text{for } (i = 0; i < N; ++i) \\
\hspace{1cm} \text{for } (j = 0; j < N - i; ++j) \\
\text{F: } a[i+j] = f(a[i+j]); \\
\text{W: Write}(a[i]);
\]

Map to all writes:

\[
R = A_2 . (A_1^{-1});
\]

Last write:

\[
\text{lexmax } R;
\]

In general: impose lexicographical order on shared iterators
Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

Simple case: array written through a single access

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        F: a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
    W: Write(a[i]);
```

Access relations:

A1:=[N]->{F[i,j]->a[i+j]:0<=i<N and 0<=j<N-i};
A2:=[N]->{W[i] -> a[i] : 0 <= i < N };
Dataflow Analysis

*Given a read from an array element, what was the last write to the same array element before the read?*

Simple case: array written through a single access

\[
\text{for } (i = 0; i < N; ++i) \\
\quad \text{for } (j = 0; j < N - i; ++j) \\
\text{F: } a[i+j] = f(a[i+j]); \\
\text{for } (i = 0; i < N; ++i) \\
\text{W: } \text{Write}(a[i]);
\]

Access relations:

\[
A1:=[N]->\{F[i,j]->a[i+j]: 0<=i<N \text{ and } 0<=j<N-i\}; \\
A2:=[N]->\{W[i] -> a[i] : 0 <= i < N \};
\]

Map to all writes: \( R := A2 \cdot (A1^\sim -1) \);
Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

Simple case: array written through a single access

```plaintext
for (i = 0; i < N; ++i)
  for (j = 0; j < N - i; ++j)
F: a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W: Write(a[i]);
```

Access relations:
- **A1**: \([N] \rightarrow \{ F[i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}\);
- **A2**: \([N] \rightarrow \{ W[i] \rightarrow a[i] : 0 \leq i < N \}\);

Map to all writes: \( R := A2 \cdot (A1^{-1}) \);

Last write: \( \text{lexmax } R \);
Dataflow Analysis

*Given a read from an array element, what was the last write to the same array element before the read?*

Simple case: array written through a single access

```plaintext
for (i = 0; i < N; ++i)
  for (j = 0; j < N - i; ++j)
    F: a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
  W: Write(a[i]);
```

Access relations:

\[ A1 : \{ F[i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} \]
\[ A2 : \{ W[i] \rightarrow a[i] : 0 \leq i < N \} \]

Map to all writes: \[ R := A2 \cdot (A1^{-1}) \]

Last write: \[ \text{lexmax } R \]

In general: impose lexicographical order on shared iterators
Dataflow Analysis

In general:

last Write before Read under Schedule

Result: last write + set of reads without corresponding write
Dataflow Analysis

In general:

last Write before Read under Schedule

Result: last write + set of reads without corresponding write

```c
for (i = 0; i < n; ++i)
    T: t[i] = a[i];
for (i = 0; i < n; ++i)
    for (j = 0; j < n - i; ++j)
        F: t[j] = f(t[j], t[j+1]);
for (i = 0; i < n; ++i)
    B: b[i] = t[i];

M := parse_file("dep.c");
Write := M[1]; Read := M[2]; Sched := M[3];
last Write before Read under Sched;
```
Transitive Closures

Given a graph (represented as an affine map)

\[
M := \{ \text{A}[i] \to \text{A}[i+1] : 0 \leq i \leq 3; \ B[] \to \text{A}[2] \};
\]

What is the transitive closure?

\[
\Rightarrow M^+;
\]

Result:

\[
\{ \text{B[]} \to \text{A}[o0] : o0 \leq 4 \text{ and } o0 \geq 3; \ \text{B[]} \to \text{A}[2]; \\
\text{A}[i] \to \text{A}[o0] : i \geq 0 \text{ and } i \leq 3 \text{ and } o0 \geq 1 \text{ and } o0 \leq 4 \text{ and } o0 \geq 1 + i \},
\]

True

exact transitive closure
Transitive Closures

Given a graph (represented as an affine map)

\[ M := \{ A[i] \rightarrow A[i+1] : 0 \leq i \leq 3; B[] \rightarrow A[2] \}; \]

What is the transitive closure? \[ \Rightarrow M^+; \]
Transitive Closures

Given a graph (represented as an affine map)

\[ M := \{ A[i] \rightarrow A[i+1] : 0 \leq i \leq 3; B[] \rightarrow A[2] \}; \]

What is the transitive closure? \[ M^+; \]

Result:

\[ \{ B[] \rightarrow A[o\emptyset] : o\emptyset \leq 4 \text{ and } o\emptyset \geq 3; B[] \rightarrow A[2]; A[i] \rightarrow A[o\emptyset] : i \geq 0 \text{ and } i \leq 3 \text{ and } o\emptyset \geq 1 \text{ and } o\emptyset \leq 4 \text{ and } o\emptyset \geq 1 + i \}, \text{ True} \]
Transitive Closures

Given a graph (represented as an affine map)

\[ M := \{ A[i] \rightarrow A[i+1] : 0 \leq i \leq 3; B[] \rightarrow A[2] \}; \]

What is the transitive closure? \( \Rightarrow M^+; \)

Result:

\[ \{ B[] \rightarrow A[o0] : o0 \leq 4 \text{ and } o0 \geq 3; B[] \rightarrow A[2]; A[i] \rightarrow A[o0] : i \geq 0 \text{ and } i \leq 3 \text{ and } o0 \geq 1 \text{ and } o0 \leq 4 \text{ and } o0 \geq 1 + i \}, \text{ True} \]
Reachability Analysis

double x[2][10];
int old = 0, new = 1, i, t;
for (t = 0; t<1000; t++) {
    for (i = 0; i<10;i++)
        x[new][i] = g(x[old][i]);
    new = (new+1) %2; old = (old+1) %2;
}

Invariant between new and old?
Reachability Analysis

```c
double x[2][10];
int old = 0, new = 1, i, t;
for (t = 0; t<1000; t++) {
    for (i = 0; i<10;i++)
        x[new][i] = g(x[old][i]);
    new = (new+1) %2; old = (old+1) %2;
}

Invariant between new and old?

T := \{[new,old] -> [(new+1)%2,(old+1)%2]\};
S0 := \{[0,1]\};
(T^+)(S0);
```
Cardinality

\[
\begin{align*}
\text{for } & (i = 0; i < N; ++i) \\
& \text{for } (j = 0; j < N - i; ++j) \\
& a[i+j] = f(a[i+j]);
\end{align*}
\]

- How many times is the statement executed?

\[
S := [N] \rightarrow \{ [i,j] : 0\leq i < N \text{ and } 0\leq j < N-i \};
\]
\[
card S;
\]
Cardinality

\[
\text{for } (i = 0; i < N; ++i) \\
\quad \text{for } (j = 0; j < N - i; ++j) \\
\quad a[i+j] = f(a[i+j]);
\]

- How many times is the statement executed?

\[
S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \};
\]

\[
\text{card } S; \quad \text{number of elements in the set}
\]
Cardinality

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  \[ S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \]

  \[ \text{card } S; \quad \text{number of elements in the set} \]

- How many times is a given array element written?

  \[ A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \]

  \[ \text{card } (A^{-1}); \]
Cardinality

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  \[ S := \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N-i \}; \]

  \[ \text{card} S; \]  
  number of elements in the set

- How many times is a given array element written?

  \[ A := \{ [i,j] -> a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N-i \}; \]

  \[ \text{card} (A^{-1}); \]  
  number of image elements
Cardinality

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?
  
  \[ S := [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \]

  \[ \text{card } S; \] number of elements in the set

- How many times is a given array element written?
  
  \[ A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \]

  \[ \text{card } (A^{-1}); \] number of image elements

- How many array elements are written?
  
  \[ A := [N] \rightarrow \{ [i,j] \rightarrow a[i+j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \}; \]

  \[ \text{card } (\text{ran } A); \]
Quasipolynomials

```plaintext
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n - 2 * i; ++j)
        /* S */
```

How many times is S executed?

card [n] -> { [i,j] : 1 <= i <= n and 1 <= j <= n - 2i };

Quasipolynomials

```cpp
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n - 2 * i; ++j)
        /* S */
```

How many times is S executed?

card \([n]\) -> \{ [i, j] : 1 <= i <= n and 1 <= j <= n - 2i \};

Result:

\([n]\) -> \{ ((-1/4 * n + 1/4 * n^2) - 1/2 * floor((n)/2)) : n >= 3 \}

That is,

\[-\frac{n}{4} + \frac{n^2}{4} - \frac{1}{2} \left\lfloor \frac{n}{2} \right\rfloor \text{ if } n \geq 3.\]
Quasipolynomials

```c
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n - 2 * i; ++j)
        /* S */
```

How many times is S executed?

card [n] -> { [i,j] : 1 <= i <= n and 1 <= j <= n - 2i };

Result:

```
[n] -> { ((-1/4 * n + 1/4 * n^2) - 1/2 * floor((n)/2)) : n >= 3 }
```

That is,

\[-\frac{n}{4} + \frac{n^2}{4} - \frac{1}{2} \left\lfloor \frac{n}{2} \right\rfloor \text{ if } n \geq 3.\]

Polynomial approximations

⇒ run iscc --polynomial-approximation
Memory Requirements

```c
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?
Memory Requirements

```c
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

\[
\text{ub } [N] \rightarrow \{[i,j] \rightarrow i\times j + i - N + 1: 0 \leq i < N \text{ and } i \leq j < N\};
\]
Memory Requirements

```c
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

\[ \text{ub } [N] \rightarrow \{[i,j] \rightarrow i\cdot j + i - N + 1: 0 \leq i < N \text{ and } i \leq j < N\}; \]

Result:

\[ ([N] \rightarrow \{ \max((1 - 2 \cdot N + N^2)) : N \geq 1 \}, \text{True}) \]
Memory Requirements

```c
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

\[
\text{ub } [N] \rightarrow \{ [i,j] \rightarrow i\times j + i - N + 1 : 0 \leq i < N \text{ and } i \leq j < N \};
\]

Result:

\[
([N] \rightarrow \{ \max((1 - 2 \times N + N^2)) : N \geq 1 \}, \text{True})
\]
Incremental Counting

```cpp
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- **direct computation**

  ```
  card [N] -> { [i,j] : 0<=i<N and 0<=j<N-i };
  ```
Incremental Counting

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- direct computation
  
  \[ \text{card } [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} ; \]

- incremental computation
  
  \[ \text{card } [N] \rightarrow \{ [i] \rightarrow [j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} ; \]
Incremental Counting

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- **direct computation**
  
  \[
  \text{card } [N] \rightarrow \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N-i \};
  \]

- **incremental computation**
  
  \[
  \text{card } [N] \rightarrow \{ [i] \rightarrow [j] : 0 \leq i < N \text{ and } 0 \leq j < N-i \};
  \]

**Result:**

\[
[N] \rightarrow \{ [i] \rightarrow (N - i) : i \leq -1 + N \text{ and } i \geq 0 \}
\]

\[
\text{sum } [N] \rightarrow \{ [i] \rightarrow (N - i) : i \leq -1 + N \text{ and } i \geq 0 \};
\]
Incremental Counting

```c
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- **direct computation**
  
  \[
  \text{card } \{ [i,j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} \]

- **incremental computation**
  
  \[
  \text{card } \{ [i] \rightarrow [j] : 0 \leq i < N \text{ and } 0 \leq j < N - i \} \]

**Result:**

\[
\{ [i] \rightarrow (N - i) : i \leq -1 + N \text{ and } i \geq 0 \}
\]

**sum** \[
\{ [i] \rightarrow (N - i) : i \leq -1 + N \text{ and } i \geq 0 \}
\]

*sum over all elements in domain*
Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?
Total Memory Allocation

\[
\text{for } (i = 0; i < N; ++i) \\n\quad \text{for } (j = i; j < N; ++j) \\n\quad \text{p}[i][j] = \text{malloc}(i * j + i - N + 1); \\n\] /* ... */
\[
\text{for } (i = 0; i < N; ++i) \\n\quad \text{for } (j = i; j < N; ++j) \\n\quad \text{free(p}[i][j]); \]

How much memory allocated in total?

\[
\text{sum} [N] \rightarrow \{[i,j] \rightarrow i*j+i-N+1: \; 0 \leq i < N \; \text{and} \; i \leq j < N\};
\]
Weighted Counting

\[ F := \{ [x, y] \rightarrow \frac{1}{4}x^2 + \frac{1}{4}y^2 : 1 \leq x, y \leq 2 \}; \]
Weighted Counting

\[ \sum_{x \in D} F(x) \]

\[ F := \{ [x,y] \rightarrow \frac{1}{4}x^2 + \frac{1}{4}y^2 : 1 \leq x,y \leq 2 \}; \]

\[ D := \text{dom} \ F; \]
Weighted Counting

\[ F := \{ [x,y] \rightarrow \frac{1}{4}x^2 + \frac{1}{4}y^2 : 1 \leq x, y \leq 2 \}; \]
\[ D := \text{dom } F; \]
\[ F(D); \]
\[ \Rightarrow \text{sum of } F \text{ over points in } D \]
Weighted Counting

\[ M : x \rightarrow (x, y) \]

\[ F := \{ [x,y] \rightarrow \frac{1}{4}x^2 + \frac{1}{4}y^2 : 1 \leq x,y \leq 2 \} ; \]
\[ D := \text{dom } F ; \]
\[ F(D) ; \]
\[ \Rightarrow \text{sum of } F \text{ over points in } D \]
\[ M := \{ [x] \rightarrow [x,y] \} ; \]

\[ \frac{x^2 + y^2}{4} \]
**Weighted Counting**

\[ M : x \rightarrow (x, y) \]

\[ F := \{ [x, y] \rightarrow \frac{1}{4}x^2 + \frac{1}{4}y^2 : 1 \leq x, y \leq 2 \}; \]

\[ D := \text{dom } F; \]

\[ F(D); \]

⇒ sum of \( F \) over points in \( D \)

\[ M := \{ [x] \rightarrow [x, y] \}; \]

\[ F(M); \]

⇒ sum of \( F \) over image of \( M \) (alternative notation: \( M \cdot F \))

\[ \sum_{i=1}^{3} \text{sum3, sum4} \]
Compositions with Piecewise (Folds of) Quasipolynomials

\( f \cdot g; \)

- \( f: D_1 \to D_2 \) is a map
- \( g: D_2 \to \mathbb{Q} \) may be
  - piecewise quasipolynomial (result of counting problems)
    \( \Rightarrow \) take sum over intersection of \( \text{ran } f \) and \( \text{dom } g \)
  - piecewise fold of quasipolynomials (result of upper bound computation)
    \( \Rightarrow \) compute bound over intersection of \( \text{ran } f \) and \( \text{dom } g \)
- \( (f \cdot g): D_1 \to \mathbb{Q} \) of same type as \( g \)

Note: if \( f \) is single-valued, then sum/bound is computed over a single point
Outline

1. Introduction

2. Basic Concepts and Operations
   - Sets and Statement Instances
   - Maps and AST Generation
   - Access Relations and Polyhedral Model
   - Dataflow Analysis
   - Transitive Closures
   - Basic Counting
   - Computing Bounds
   - Weighted Counting

3. Simple Applications
   - Pointer Conversion
   - Dynamic Memory Requirement Estimation
   - Reuse Distance Computation
### Pointer Conversion

Let's consider the following code:

```c
p = a;
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p += 1 + j * ((j-i)/4);
        *p = hard_work(i,j);
    }
```

Can we parallelize this code?
Pointer Conversion

```c
p = a;
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p += 1 + j * ((j-i)/4);
        *p = hard_work(i,j);
    }
```

Can we parallelize this code?

⇒ No, (false) dependency through `p`
⇒ Compute closed formula for `p`

\[
p = a + \sum_{(i', j') \in S} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor
\]

with \(S = \{ (i', j') \in \mathbb{Z}^2 | 0 \leq i' < N \land i' \leq j' < N \}\)
Pointer Conversion

\[ p = a; \]
\[ \text{for } (i = 0; i < N; ++i) \]
\[ \quad \text{for } (j = i; j < N; ++j) \{ \]
\[ \quad \quad p += 1 + j * ((j-i)/4); \]
\[ \quad \quad *p = \text{hard\_work}(i,j); \]
\[ \} \]

Can we parallelize this code?

⇒ No, (false) dependency through \( p \)
⇒ Compute closed formula for \( p \)

\[ p = a + \sum_{(i',j') \in S} j' \left\lceil \frac{j' - i'}{4} \right\rceil \]

with \( S = \{ (i',j') \in \mathbb{Z}^2 \mid 0 \leq i' < N \land i' \leq j' < N \} \)

lexicographically less than
Pointer Conversion

\[ p = a + \sum_{(i',j') \in S} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor \]

with \( S = \{(i', j') \in \mathbb{Z}^2 \mid 0 \leq i' < N \land i' \leq j' < N\} \)
Pointer Conversion

\[ p = a + \sum_{(i',j') \in S} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor \]

with \( S = \{(i',j') \in \mathbb{Z}^2 | 0 \leq i' < N \land i' \leq j' < N\} \)

\[
S := \{[i,j] : 0 \leq i < N \land i \leq j < N \} ;
\]

\[
L := S \ll= S ;
\]

\[
INC := \{ [[i,j] \to [i',j']] \to 1 + j' * \lfloor (j'-i')/4 \rfloor \} ;
\]

\[
INC := INC * (\text{wrap} \ (L^\text{-1})) ;
\]

\[
\text{sum} \ INC ;
\]
Pointer Conversion

\[ p = a + \sum_{(i',j') \in S} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor \]

with \( S = \{ (i', j') \in \mathbb{Z}^2 \mid 0 \leq i' < N \land i' \leq j' < N \} \)

map: (elements of) left set lexicographically smaller than right set

\[
\text{S := } [N] \rightarrow \{ [i, j] : 0 \leq i < N \text{ and } i \leq j < N \}; \text{L := S <<= S; INC := \{ [[i, j] \rightarrow [i', j']] \rightarrow 1 + j' * \left\lfloor (j' - i')/4 \right\rfloor \}; INC := INC * (\text{wrap} (\text{L}^{-1})); \text{sum INC;}
\]
Pointer Conversion

\[ p = a + \sum_{(i', j') \in S} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor \]

with \( S = \{(i', j') \in \mathbb{Z}^2 \mid 0 \leq i' < N \land i' \leq j' < N\} \)

map: (elements of) left set lexicographically smaller than right set

\[
S := [N] \rightarrow \{ [i, j] : 0 \leq i < N \land i \leq j < N \}; \\
L := S \ll S; \\
INC := \{ [[i, j] \rightarrow [i', j']] \rightarrow 1 + j' \times \left\lfloor (j' - i')/4 \right\rfloor \}; \\
INC := INC \times (\text{wrap} \ (L^\sim 1)); \\
\text{sum} \ INC;
\]

embed map in a set
Pointer Conversion

\[ p = a + \sum_{(i', j') \in S \atop (i', j') \preceq (i, j)} j' \left\lfloor \frac{j' - i'}{4} \right\rfloor \]

with \( S = \{(i', j') \in \mathbb{Z}^2 \mid 0 \leq i' < N \land i' \leq j' < N\} \)

map: (elements of) left set lexicographically smaller than right set

\[
S := \mathbb{N} \rightarrow \{ [i, j] : 0 \leq i < N \land i \leq j < N \};
\]

\[
L := S \ll S;
\]

\[
INC := \{ [[i, j] \rightarrow [i', j']] \rightarrow 1 + j' \star \left(\frac{j' - i'}{4}\right)\};
\]

\[
INC := INC \star (\text{wrap} (L^\star 1));
\]

\[
\text{sum} \ INC;
\]

Note: if domain of argument to \text{sum} [ub] is an embedded map, then \text{sum} [bound] is computed over range of embedded map
Dynamic Memory Requirement Estimation [CFGV2006]

How much memory is needed to execute the following program?

```cpp
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```
Dynamic Memory Requirement Estimation [CFGV2006]

How much memory is needed to execute the following program?

```c
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /*S1*/
        B[] m2Arr = m2(2*m-c); /*S2*/
    }
}

void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /*S3*/
        B[] dummyArr = m2(i); /*S4*/
    }
}

B[] m2(int n) {
    B[] arrB = new B[n]; /*S5*/
    for (j = 1; j <= n; j++)
        B b = new B(); /*S6*/
    return arrB;
}
```

D := {
    m0[m]→S1[c] : 0<=c<m;
    m0[m]→S2[c] : 0<=c<m;
    m1[k]→S3[i] : 1<=i<=k;
    m1[k]→S4[i] : 1<=i<=k;
    m2[n]→S5[];
    m2[n]→S6[j] : 1<=j<=n
};
Dynamic Memory Requirement Estimation [CFGV2006]

How much (scoped) memory is needed?

⇒ compute for each method

\[ \text{ret}_m \] size of memory returned by \( m \)

\[ \text{cap}_m \] size of memory “captured” (not returned) by \( m \)

\[ \text{memRq}_m \] total memory requirements of \( m \)

\[ \text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p \]

\[ \text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p \]
Dynamic Memory Requirement Estimation [CFGV2006]

How much (scoped) memory is needed?

⇒ compute for each method

\[ \text{ret}_m \text{ size of memory returned by } m \]

\[ \text{cap}_m \text{ size of memory “captured” (not returned) by } m \]

\[ \text{memRq}_m \text{ total memory requirements of } m \]

\[ \text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p \]

\[ \text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p \]

⇒ summarize over statement instances, i.e., compose with

\[ M = (\text{dom } l)^{-1} \]

\[ D := \{ \]}

\[ m0[m] \rightarrow S1[c] : 0 \leq c < m; \]

\[ m0[m] \rightarrow S2[c] : 0 \leq c < m; \]

\[ m1[k] \rightarrow S3[i] : 1 \leq i \leq k; \]

\[ m1[k] \rightarrow S4[i] : 1 \leq i \leq k; \]

\[ m2[n] \rightarrow S5[ ]; \]

\[ m2[n] \rightarrow S6[j] : 1 \leq j \leq n \}; \]

\[ \text{DM} := (\text{domain_map } D)^{-1}; \]
Dynamic Memory Requirement Estimation [CFGV2006]

How much (scoped) memory is needed?
⇒ compute for each method

- \( \text{ret}_m \) size of memory returned by \( m \)
- \( \text{cap}_m \) size of memory “captured” (not returned) by \( m \)
- \( \text{memRq}_m \) total memory requirements of \( m \)

\[
\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p
\]

\[
\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p
\]
Dynamic Memory Requirement Estimation [CFGV2006]

How much (scoped) memory is needed?
⇒ compute for each method

- \( \text{ret}_m \) size of memory returned by \( m \)
- \( \text{cap}_m \) size of memory “captured” (not returned) by \( m \)
- \( \text{memRq}_m \) total memory requirements of \( m \)

\[
\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p
\]

\[
\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p
\]

```java
B[] m2(int n) {
    B[] arrB = new B[n];
    for (j=1; j<=n; j++)
        B b = new B();
    return arrB;
}
```
Dynamic Memory Requirement Estimation [CFGV2006]

How much (scoped) memory is needed?
⇒ compute for each method

\[
\text{ret}_m \quad \text{size of memory returned by } m \\
\text{cap}_m \quad \text{size of memory "captured" (not returned) by } m \\
\text{memRq}_m \quad \text{total memory requirements of } m
\]

\[
\text{ret}_m + \text{cap}_m = \sum_{p \text{ called by } m} \text{ret}_p
\]

\[
\text{memRq}_m = \text{cap}_m + \max_{p \text{ called by } m} \text{memRq}_p
\]

B[] m2(int n) {
    B[] arrB = new B[n];
    for (j=1; j<=n; j++)
        B b = new B();
    return arrB;
}

ret_m2 := DM .
    { [m2[n] -> S5[]] -> n : n >= 0 };

cap_m2 := DM .
    { [m2[n] -> S6[j]] -> 1 };

req_m2 := cap_m2 +
    { m2[n] -> max(0) };

mem2
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A(); /* S3 */
        B[] dummyArr = m2(i); /* S4 */
    }
}

cap_{m1}(k) = \sum_{1 \leq i \leq k} (1 + ret_{m2}(i))

ret_{m2} is a function of the arguments of m2
We want to use it as a function of the arguments and local variables of m1
Dynamic Memory Requirement Estimation [CFGV2006]

```c
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();    /* S3 */
        B[] dummyArr = m2(i);    /* S4 */
    }
}
```

\[
\text{cap}_{m1}(k) = \sum_{1 \leq i \leq k} (1 + \text{ret}_{m2}(i))
\]

\text{ret}_m2 is a function of the arguments of \text{m2}

We want to use it as a function of the arguments and local variables of \text{m1}

\[\Rightarrow\text{define parameter binding}\]

\[\text{CB}_m1 := \{ [m1[k] \rightarrow S4[i]] \rightarrow m2[i] \} ;\]

\[\text{cap}_m1 := \text{DM} . (\{ [m1[k] \rightarrow S3[i]] \rightarrow 1 \} + (\text{CB}_m1 . \text{ret}_m2)) ;\]
Dynamic Memory Requirement Estimation [CFGV2006]

```java
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();    /* S3 */
        B[] dummyArr = m2(i);    /* S4 */
    }
}
```

\[
\text{memRq}_m = \text{cap}_m + \max_{\text{p \ called by } m}\ \text{memRq}_p
\]

\[
\begin{align*}
\text{CB}_m1 & := \{ [m1[k] -> S4[i]] -> m2[i] \}; \\
\text{ret}_m1 & := \{ m1[k] -> 0 \}; \\
\text{cap}_m1 & := \text{DM} \cdot (\{ [m1[k]->S3[i]] -> 1 \} + (\text{CB}_m1 \cdot \text{ret}_m2)); \\
\text{req}_m1 & := \text{cap}_m1 + (\text{DM} \cdot \text{CB}_m1 \cdot \text{req}_m2);
\end{align*}
\]
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /* S1 */
        B[] m2Arr = m2(2 * m - c); /* S2 */
    }
}

CB_m0 := \{[m0[m] -> S1[c]] -> m1[c];
            [m0[m] -> S2[c]] -> m2[2 * m - c] \};
ret_m0 := \{m0[m] -> 0 \};
cap_m0 := DM . CB_m0 . (ret_m1 + ret_m2);
req_m0 := cap_m0 + (DM . CB_m0 . (req_m1 . req_m2));
Dynamic Memory Requirement Estimation [CFGV2006]

```c
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c); /* S1 */
        B[] m2Arr = m2(2 * m - c); /* S2 */
    }
}
```

\[
\begin{align*}
    CB_{m0} & := \{ [m0[m] \rightarrow S1[c]] \rightarrow m1[c]; \\
         & \hspace{1cm} [m0[m] \rightarrow S2[c]] \rightarrow m2[2 * m - c] \} ; \\
    ret_{m0} & := \{ m0[m] \rightarrow 0 \} ; \\
    cap_{m0} & := DM . CB_{m0} . (ret_{m1} + ret_{m2}) ; \\
    req_{m0} & := cap_{m0} + (DM . CB_{m0} . (req_{m1} + req_{m2})) ;
\end{align*}
\]

combine reductions
**Reuse Distance Computation**

Given an access to a cache line $\ell$, how many distinct cache lines have been accessed since the previous access to $\ell$?

⇒ Is the cache line still in the cache?

<table>
<thead>
<tr>
<th>$i$</th>
<th>Accesses</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>$c$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
**Reuse Distance Computation**

Given an access to a cache line \( \ell \), how many distinct cache lines have been accessed since the previous access to \( \ell \)?

\[ \Rightarrow \text{Is the cache line still in the cache?} \]

\begin{verbatim}
for (i = 0; i <= 7; ++i) {
    A[i];       //reference a
    A[7-i];     //reference b
    if (i <= 3)
        A[2*i];    //reference c
}
\end{verbatim}

Assume \( A[i] \) in cache line \( [i/3] \)
Reuse Distance Computation

Given an access to a cache line \( \ell \), how many distinct cache lines have been accessed since the previous access to \( \ell \)?

\[ \Rightarrow \text{Is the cache line still in the cache?} \]

```c
for (i = 0; i <= 7; ++i) {
    A[i]; //reference a
    A[7-i]; //reference b
    if (i <= 3)
        A[2*i]; //reference c
}
```

Assume \( A[i] \) in cache line \([i/3]\)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>( r@i )</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>([r@i]/3)</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>distance</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Reuse Distance Computation

```c
for (i = 0; i <= 7; ++i) {
    A[i];  //reference a
    A[7-i];  //reference b
    if (i <= 3)
        A[2*i];  //reference c
}
```

Assume A[i] in cache line [i/3]
Reuse Distance Computation

```c
for (i = 0; i <= 7; ++i) {
    A[i]; //reference a
    A[7-i]; //reference b
    if (i <= 3)
        A[2*i]; //reference c
}
```

Assume A[i] in cache line ⌊i/3⌋

D := \{ a[i] : 0 ≤ i ≤ 7; b[i] : 0 ≤ i ≤ 7; c[i] : 0 ≤ i ≤ 3 \};
C := \{ A[i] -> L[j] : j = \text{floor}(i/3) \};
S := \{ a[i] -> [i,0]; b[i] -> [i,1]; c[i] -> [i,2] \} * D;
```
Reuse Distance Computation

\[
\text{for } (i = 0; i \leq 7; ++i) \{ \\
\quad A[i]; \quad // \text{reference } a \\
\quad A[7-i]; \quad // \text{reference } b \\
\quad \textbf{if } (i \leq 3) \\
\quad \quad A[2*i]; \quad // \text{reference } c \\
\}\]

Assume \(A[i]\) in cache line \([i/3]\)

\[
D := \{ a[i] : 0 \leq i \leq 7; b[i] : 0 \leq i \leq 7; c[i] : 0 \leq i \leq 3 \}; \\
C := \{ A[i] -> L[j] : j = \text{floor}(i/3) \}; \\
S := \{ a[i] -> [i,0]; b[i] -> [i,1]; c[i] -> [i,2] \} \cdot D; \\
TIME := \text{ran } S; \quad LT := \text{TIME} \ll \text{TIME}; \quad LE := \text{TIME} \ll\ll \text{TIME}; \\
T := ((S^\neg1) . A . (A^\neg1) . S) \cdot LT; \\
M := \text{lexmin } T; \\
NEXT := S . M . (S^\neg1); \quad \# \text{ map to next access to same cache line} \\
AFTER\_PREV := (NEXT^\neg1) . (S . LE . (S^\neg1)); \\
BEFORE := S . (LE^\neg1) . (S^\neg1); \\
card ((AFTER\_PREV \cdot BEFORE) . A);
\]