Major Research Initiatives in GCC Resource Center

Uday Khedker

(www.cse.iitb.ac.in/~uday)

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

Oct 2008

Part 1

Introduction

Broad objectives

Theoretical research supported by empirical evidence

- Exploring research issues in real compilers
- Demonstrating the relevance and effectiveness (of our explorations) in real compilers

Broad Areas of Interests

- Program Analysis and Optimization
- Translation Validation
- Retargetable compilation
- Parallelization and Vectorization for SIMD and MIMD Architectures

General explorations applied in the context of GCC

Examples of Research Commitments

- Interprocedural data flow analysis
- Heap reference analysis
- Static inferencing of flow sensitive polymorphic types
- Translation validation of GCC generated code
- Increasing trustworthiness of GCC
    - Cleaner machine descriptions for GCC
    - Generating GCC optimizers from specifications

Part 2

Interprocedural Data Flow Analysis

Interprocedural Data Flow Analysis [CC2008]

- Objectives:
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:

Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge:
- The State of Art:
- Our Breakthrough:
- The Consequences:

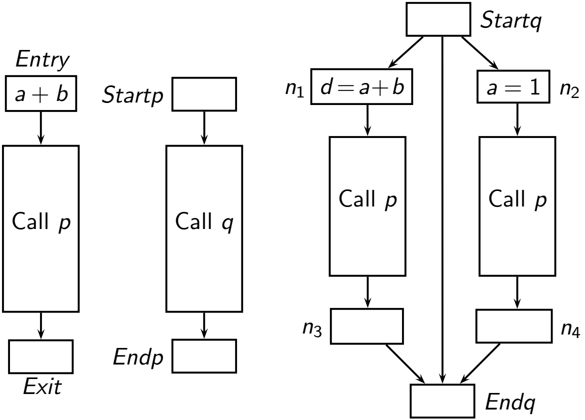
Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art:
- Our Breakthrough:
- The Consequences:

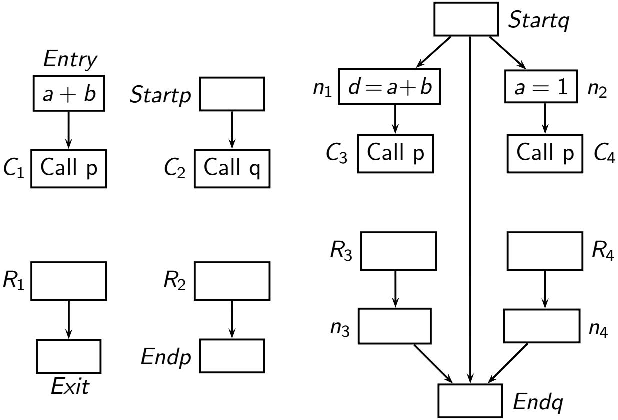
Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art: Merge information across contexts for efficiency.
  $\Rightarrow$ Significant imprecision in recursive programs.
- Our Breakthrough:
- The Consequences:

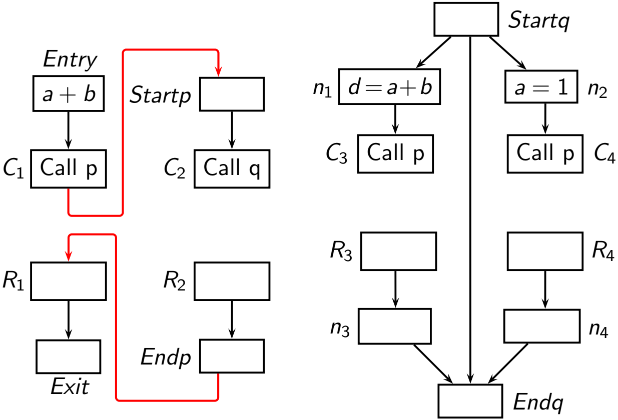
Defining Interprocedural Context for Static Analysis



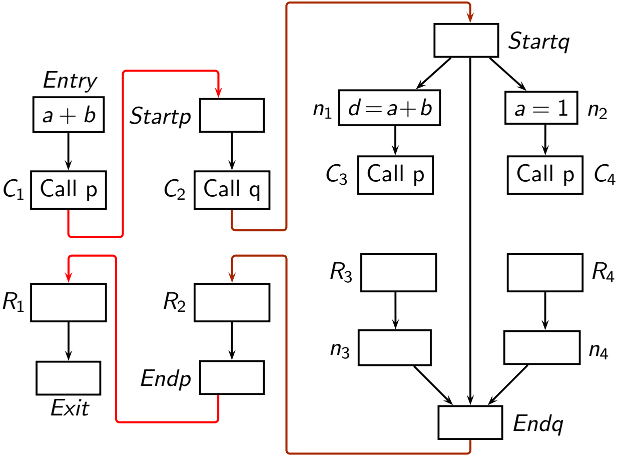
Defining Interprocedural Context for Static Analysis



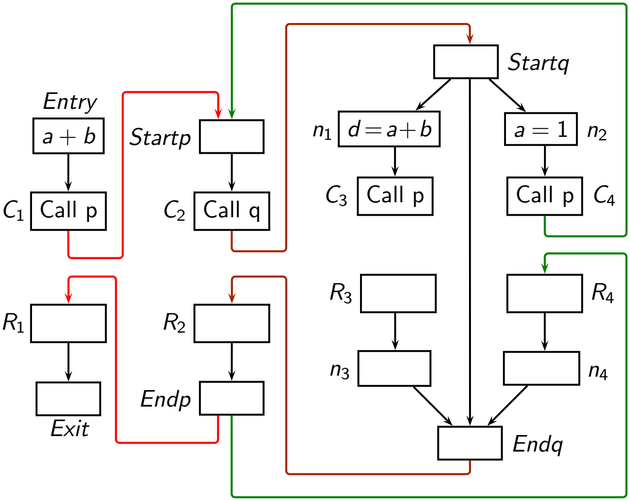
Defining Interprocedural Context for Static Analysis



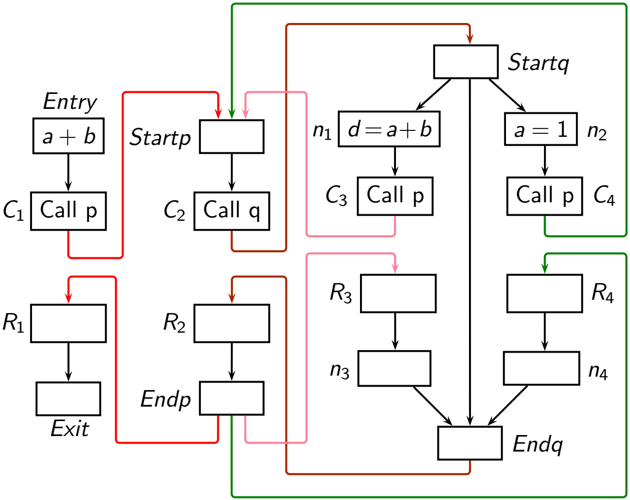
Defining Interprocedural Context for Static Analysis



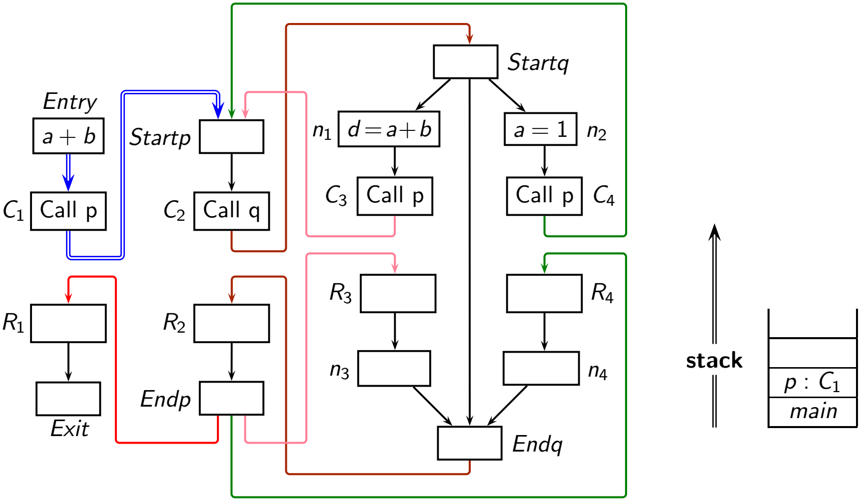
Defining Interprocedural Context for Static Analysis



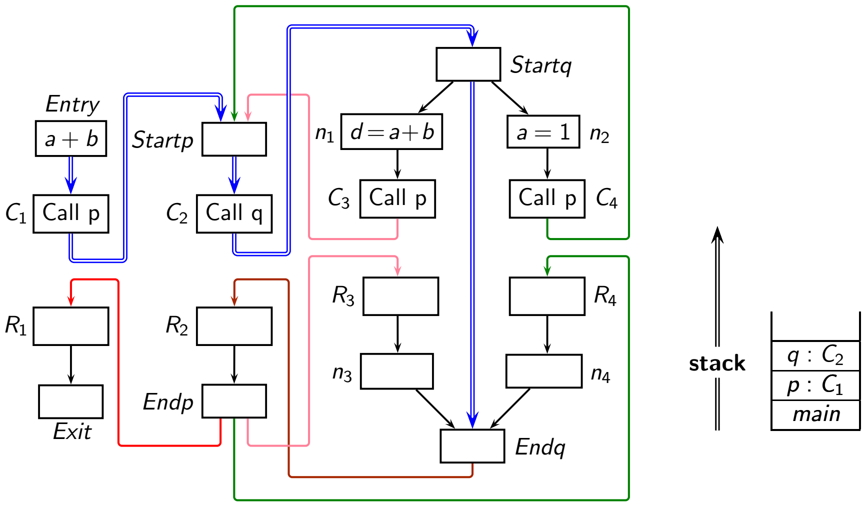
Defining Interprocedural Context for Static Analysis



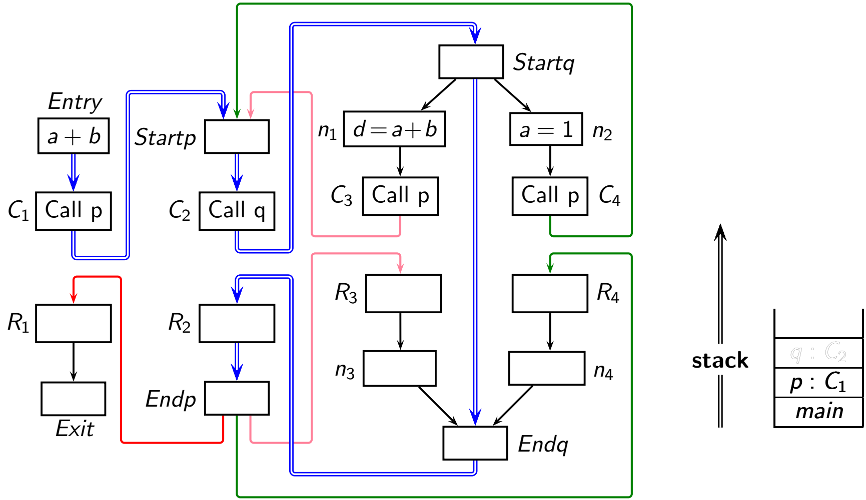
Defining Interprocedural Context for Static Analysis



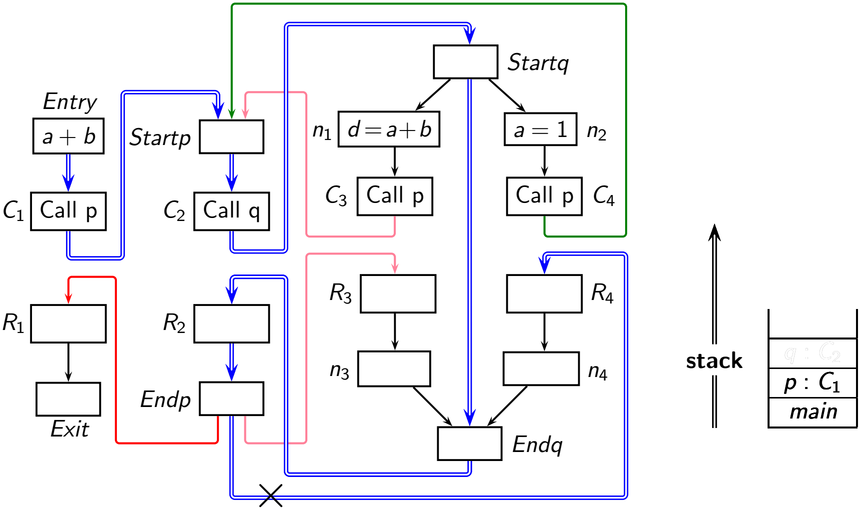
Defining Interprocedural Context for Static Analysis



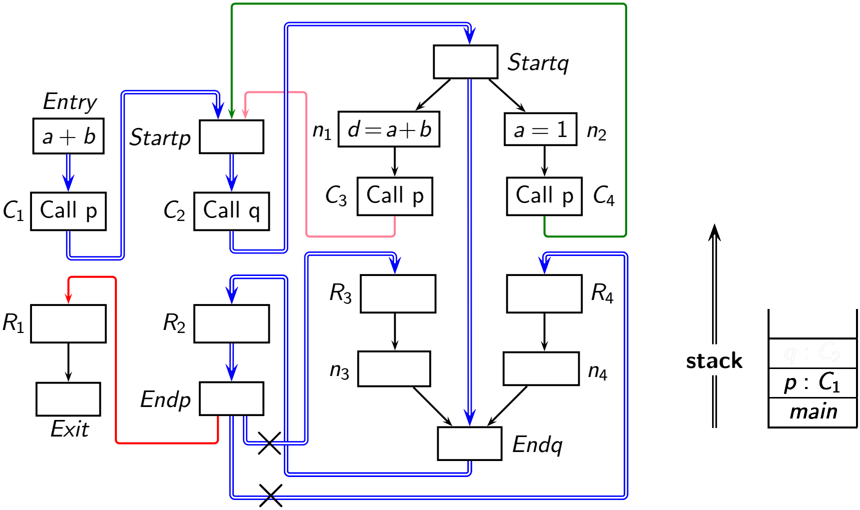
Defining Interprocedural Context for Static Analysis



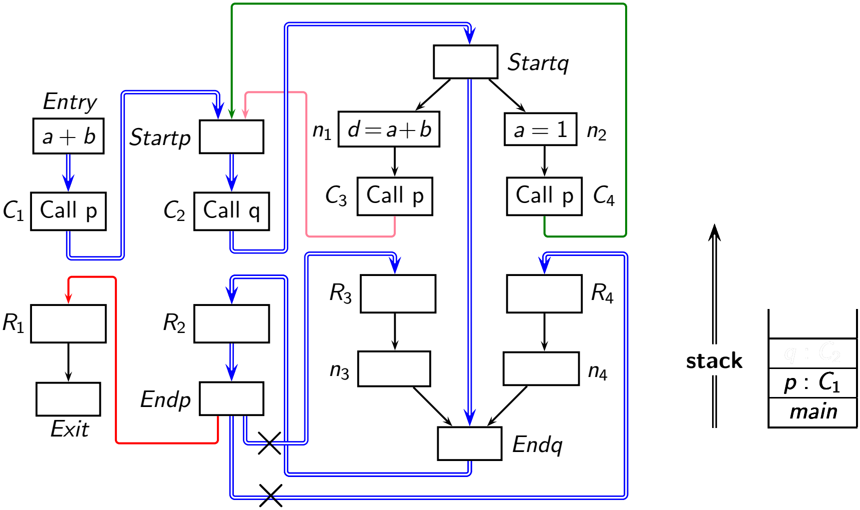
Defining Interprocedural Context for Static Analysis



Defining Interprocedural Context for Static Analysis



Defining Interprocedural Context for Static Analysis



Context is defined by stack snapshot $\Rightarrow$ Unbounded number of contexts

Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art: Merge information across contexts for efficiency.
  $\Rightarrow$ Significant imprecision in recursive programs.
- Our Breakthrough:
- The Consequences:

Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art: Merge information across contexts for efficiency.
  $\Rightarrow$ Significant imprecision in recursive programs.
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure.
- The Consequences:

Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art: Merge information across contexts for efficiency.
  $\Rightarrow$ Significant imprecision in recursive programs.
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure.
- The Consequences: Our implementation in GCC shows time and space savings by

Interprocedural Data Flow Analysis [CC2008]

- Objectives: Optimizations across procedure boundaries to incorporate
  - the effects of procedure calls in the caller procedures, and
  - the effects of calling contexts in the callee procedures.
- Main Challenge: Precision requires distinguishing between an impractically large number ($>>$ millions) of contexts at each program point.
- The State of Art: Merge information across contexts for efficiency.
  $\Rightarrow$ Significant imprecision in recursive programs.
- Our Breakthrough: Clean, formally provable characterizations to
  - discard redundant contexts at the start of every procedure, and
  - simulate regeneration contexts at the end of every procedure.
- The Consequences: Our implementation in GCC shows time and space savings by a factor of over a million!

Part 3

Heap Reference Analysis

Heap Reference Analysis [TOPLAS 2007]

- The Problem:
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives:
- Main Challenge:
- Our Key Idea:
- Current status:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge:
- Our Key Idea:
- Current status:

Heap Reference Analysis [TOPLAS 2007]

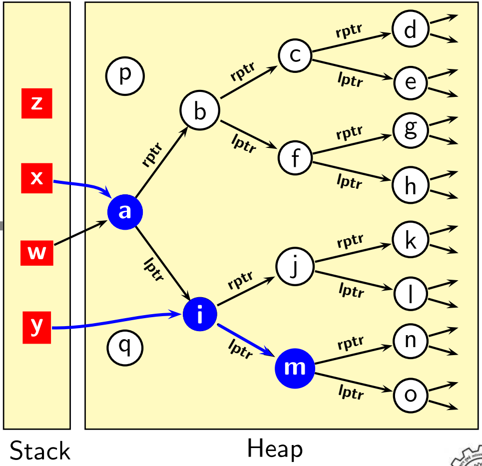
- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea:
- Current status:

Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is not executed even once.

```
w = x          // x points to m_a
while  (x.data < max)
       x = x.rptr
y = x.lptr

z = New  class_of_z
y = y.lptr
z.sum = x.data + y.data
```

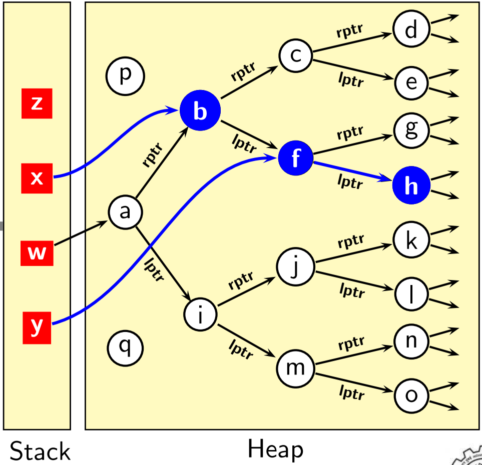


Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed once.

```
w = x          // x points to m_a
while  (x.data < max)
        x = x.rptr
y = x.lptr

z = New  class_of_z
y = y.lptr
z.sum = x.data + y.data
```

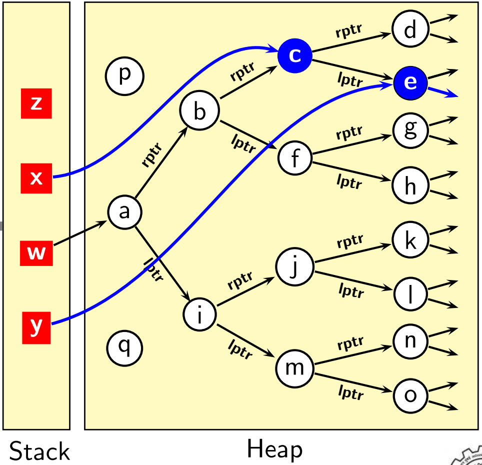


Which Heap Memory Nodes Can be Statically Marked as Live?

If the while loop is executed twice.

```
w = x          // x points to m_a
while  (x.data < max)
        x = x.rptr
y = x.lptr

z = New  class_of_z
y = y.lptr
z.sum = x.data + y.data
```



Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea:
- Current status:
- Future Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea: Represent abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status:
- Future Work:

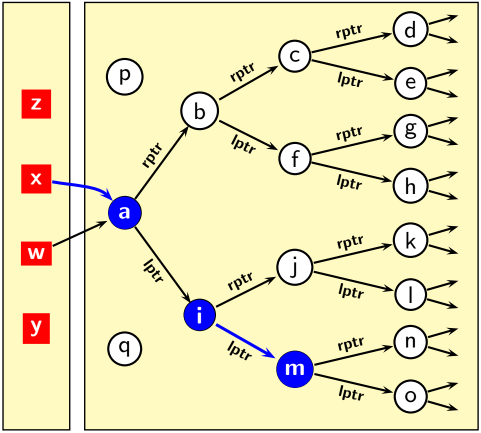
Our Solution

		y = z = null
1	w = x	
		w = null
2	while (x.data < max)	
	{	x.lptr = null
3	x = x.rptr }	
		x.rptr = x.lptr.rptr = null
		x.lptr.lptr.lptr = null
		x.lptr.lptr.rptr = null
4	y = x.lptr	
		x.lptr = y.rptr = null
		y.lptr.lptr = y.lptr.rptr = null
5	z = *New class_of_z*	
		z.lptr = z.rptr = null
6	y = y.lptr	
		y.lptr = y.rptr = null
7	z.sum = x.data + y.data	
		x = y = z = null

Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

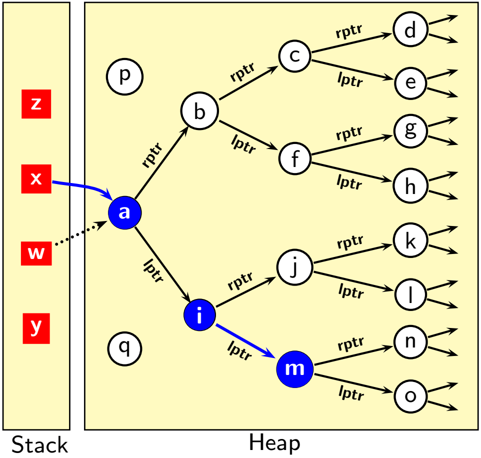
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
  y = z = null
1 w = x
  w = null
2 while (x.data < max)
  {      x.lptr = null
3        x = x.rptr        }
  x.rptr = x.lptr.rptr = null
  x.lptr.lptr.lptr = null
  x.lptr.lptr.rptr = null
4 y = x.lptr
  x.lptr = y.rptr = null
  y.lptr.lptr = y.lptr.rptr = null
5 z = New  class_of_z
  z.lptr = z.rptr = null
6 y = y.lptr
  y.lptr = y.rptr = null
7 z.sum = x.data + y.data
  x = y = z = null
```

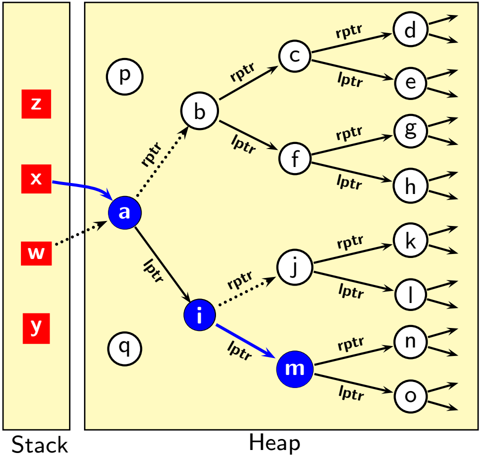
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

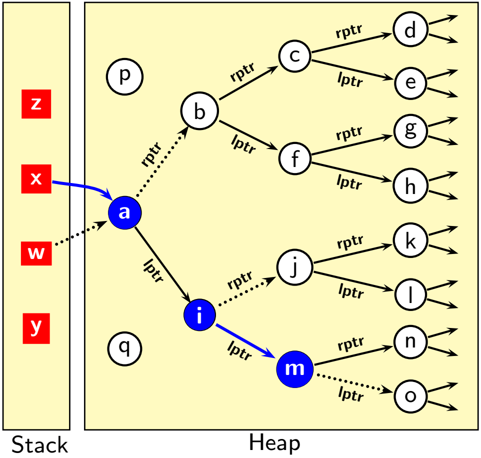
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr        }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

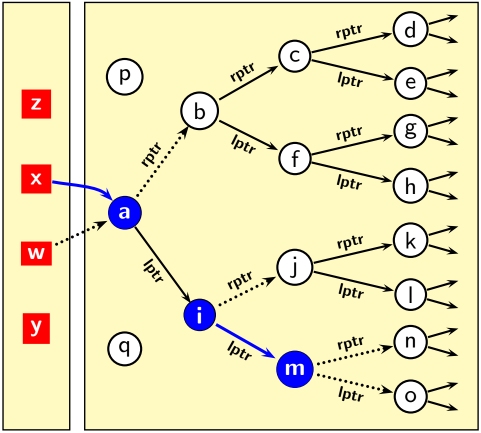
While loop is not executed even once



Heap Reference Analysis: Our Solution

y = z = null
1 w = x
w = null
2 while (x.data < max)
{ x.lptr = null
3 x = x.rptr }
x.rptr = x.lptr.rptr = null
x.lptr.lptr.lptr = null
x.lptr.lptr.rptr = null
4 y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
5 z = *New class_of_z*
z.lptr = z.rptr = null
6 y = y.lptr
y.lptr = y.rptr = null
7 z.sum = x.data + y.data
x = y = z = null

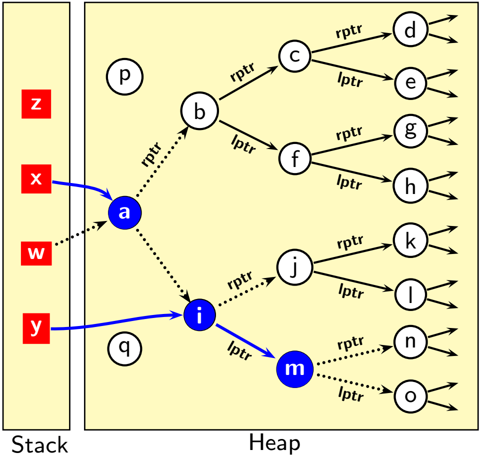
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

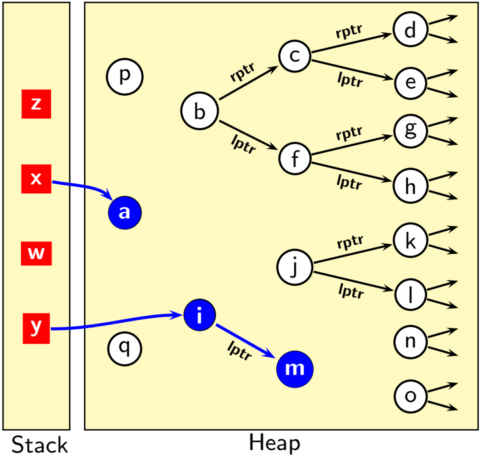
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
     y = z = null
1    w = x
     w = null
2    while (x.data < max)
     {       x.lptr = null
3            x = x.rptr       }
     x.rptr = x.lptr.rptr = null
     x.lptr.lptr.lptr = null
     x.lptr.lptr.rptr = null
4    y = x.lptr
     x.lptr = y.rptr = null
     y.lptr.lptr = y.lptr.rptr = null
5    z = New  class_of_z
     z.lptr = z.rptr = null
6    y = y.lptr
     y.lptr = y.rptr = null
7    z.sum = x.data + y.data
     x = y = z = null
```

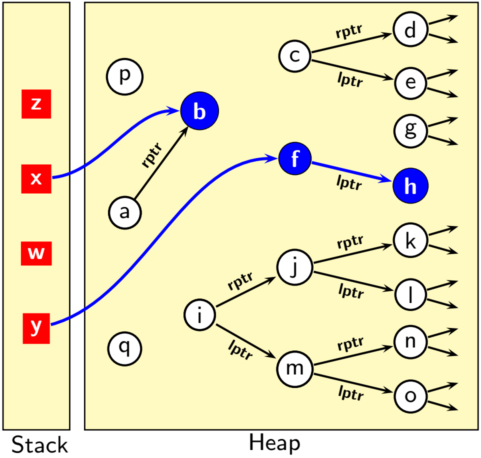
While loop is not executed even once



Heap Reference Analysis: Our Solution

```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {     x.lptr = null
3         x = x.rptr     }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

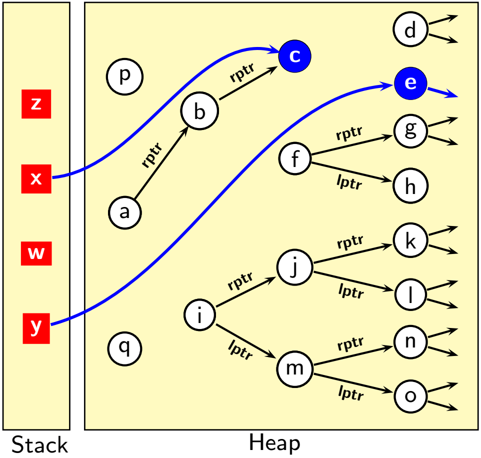
While loop is executed once



Heap Reference Analysis: Our Solution

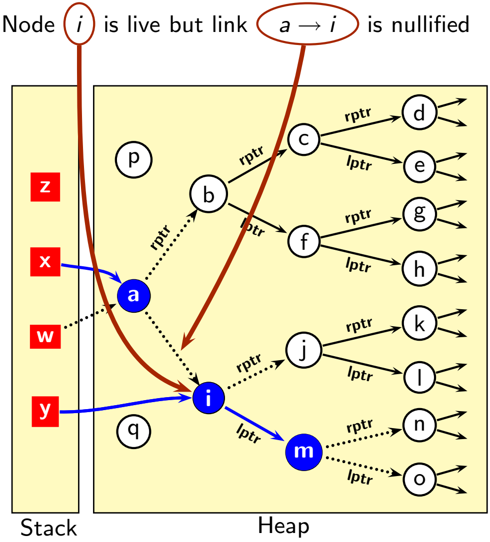
```
    y = z = null
1   w = x
    w = null
2   while (x.data < max)
    {       x.lptr = null
3           x = x.rptr       }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
4   y = x.lptr
    x.lptr = y.rptr = null
    y.lptr.lptr = y.lptr.rptr = null
5   z = New  class_of_z
    z.lptr = z.rptr = null
6   y = y.lptr
    y.lptr = y.rptr = null
7   z.sum = x.data + y.data
    x = y = z = null
```

While loop is executed twice



Some Observations

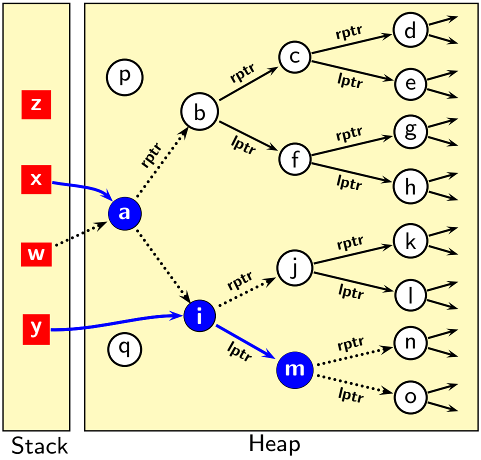
```
  y = z = null
1 w = x
  w = null
2 while (x.data < max)
  {     x.lptr = null
3       x = x.rptr       }
  x.rptr = x.lptr.rptr = null
  x.lptr.lptr.lptr = null
  x.lptr.lptr.rptr = null
4 y = x.lptr
  x.lptr = y.rptr = null
  y.lptr.lptr = y.lptr.rptr = null
5 z = New  class_of_z
  z.lptr = z.rptr = null
6 y = y.lptr
  y.lptr = y.rptr = null
7 z.sum = x.data + y.data
  x = y = z = null
```



Some Observations

```
     y = z = null
1    w = x
     w = null
2    while (x.data < max)
     {      x.lptr = null
3           x = x.rptr      }
     x.rptr = x.lptr.rptr = null
     x.lptr.lptr.lptr = null
     x.lptr.lptr.rptr = null
4    y = x.lptr
     x.lptr = y.rptr = null
     y.lptr.lptr = y.lptr.rptr = null
5    z = New class_of_z
     z.lptr = z.rptr = null
6    y = y.lptr
     y.lptr = y.rptr = null
7    z.sum = x.data + y.data
     x = y = z = null
```

New access expressions are created.
Can they cause exceptions?



Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea: Represent abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status:
- Future Work:

Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea: Represent abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Future Work:

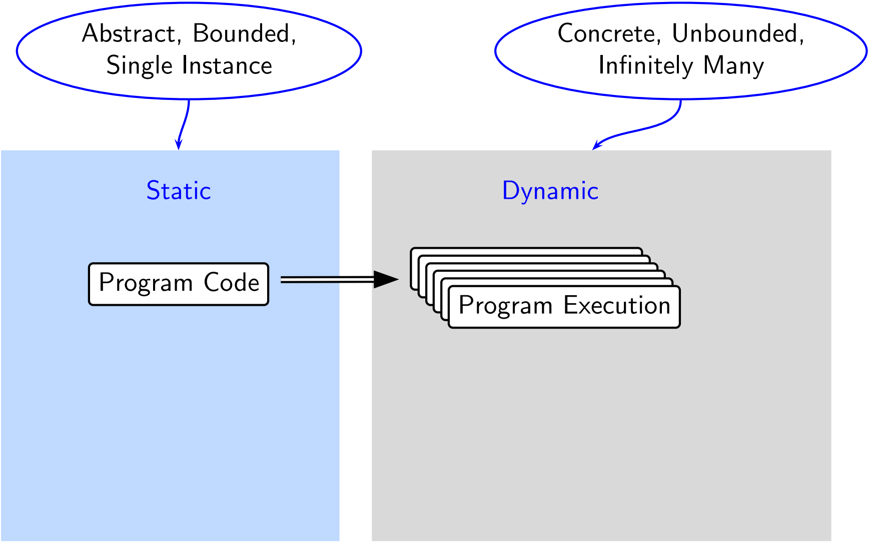
Heap Reference Analysis [TOPLAS 2007]

- The Problem: A lot of unused data remains unclaimed even in the best of garbage collectors. In C/C++, memory leaks is a major problem.
- Our Objectives: To perform static analysis of heap allocated data for making unused data unreachable in order to improve garbage collection and plug memory leaks.
- Main Challenge: Unlike stack and static data, the mapping between object names and their addresses keeps changing at runtime for heap data.
- Our Key Idea: Represent abstractions of heap data in terms of graphs and perform analysis using these graphs as data flow values.
- Current status: Theory and prototype implementation (at the intraprocedural level) ready for Java.
- Future Work:
  - Analysis for functional languages
  - Interprocedural implementation and Performance tuning
  - Implementation for C++

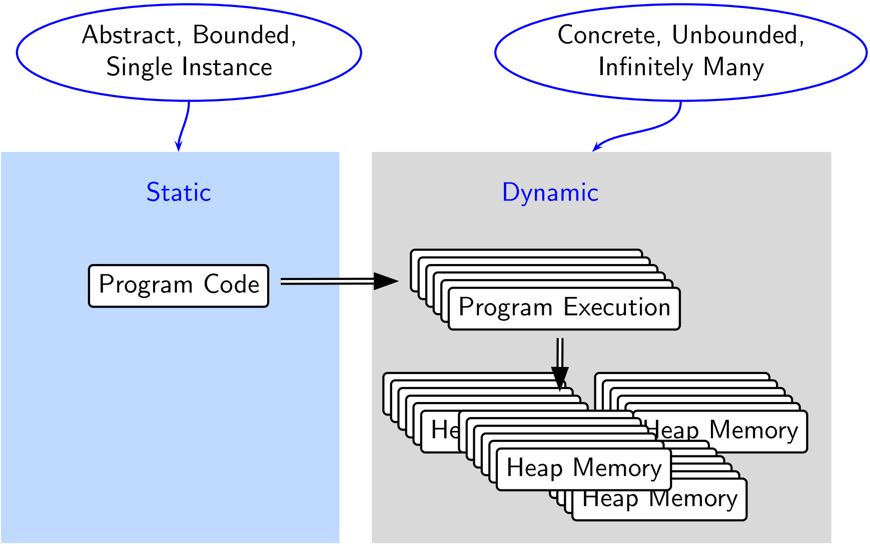
BTW, What is Static Analysis of Heap?

Static	Dynamic

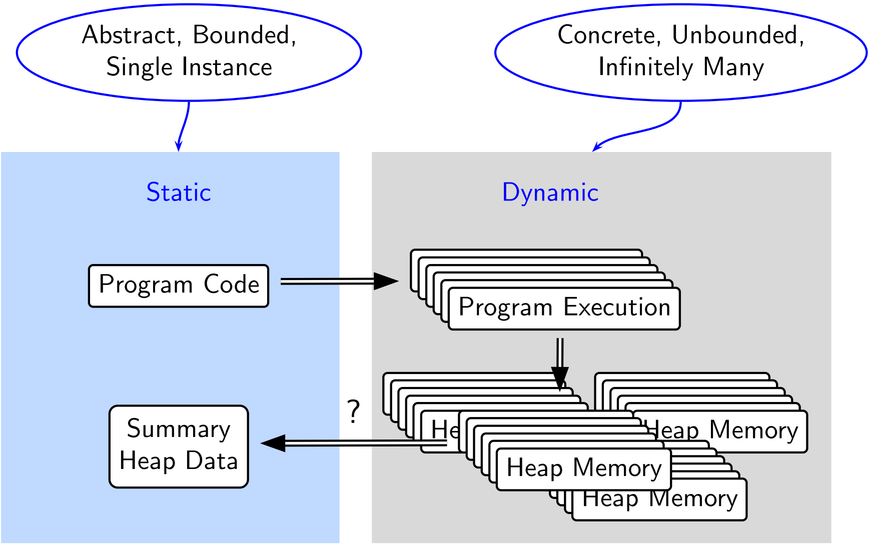
BTW, What is Static Analysis of Heap?



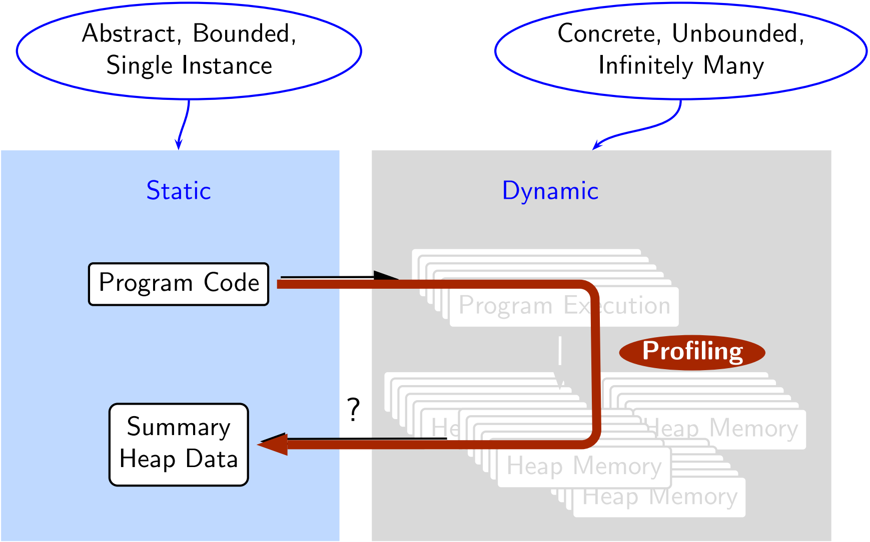
BTW, What is Static Analysis of Heap?



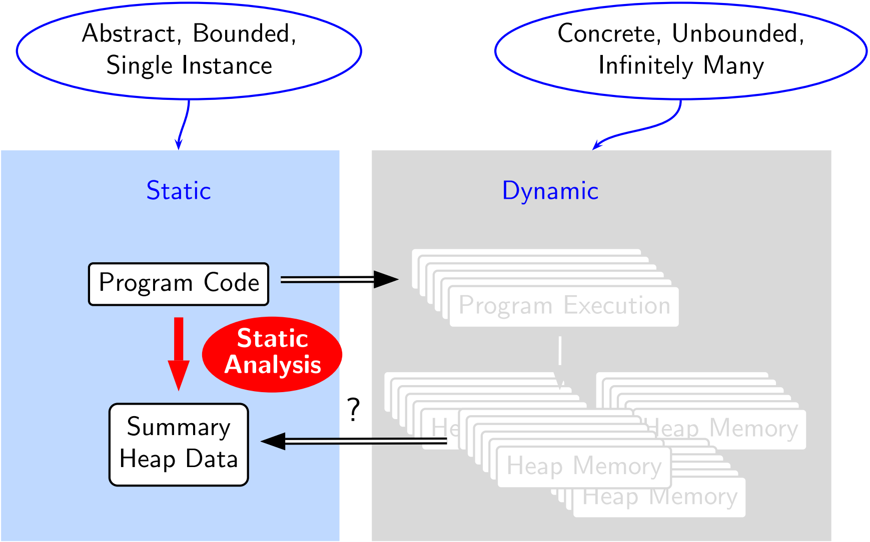
BTW, What is Static Analysis of Heap?



BTW, What is Static Analysis of Heap?



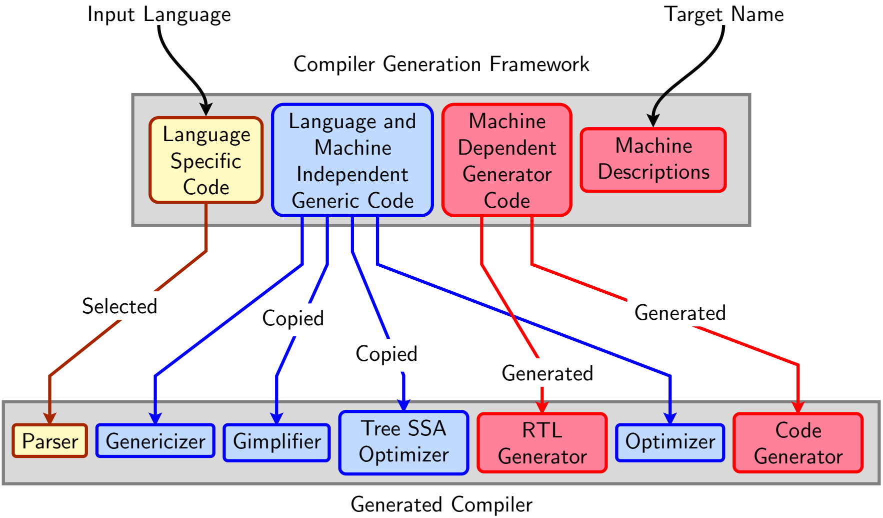
BTW, What is Static Analysis of Heap?



Part 4

Improving Instruction Selection in GCC

GCC : The GNU Compiler Collection



Improving Retargetability and Instruction Selection in GCC

- The Problem:
- The Consequences:

Improving Retargetability and Instruction Selection in GCC

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:

Improving Retargetability and Instruction Selection in GCC

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.

Improving Retargetability and Instruction Selection in GCC

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.
  - The machine descriptions are difficult to construct, understand, maintain, and enhance.

Improving Retargetability and Instruction Selection in GCC

- The Problem: Instruction selection algorithms in GCC are very primitive (employ full tree matching instead of tree tiling).
- The Consequences:
  - A compiler developer needs to visualize and specify meaningful combinations of instructions for generating good quality code.
  - The machine descriptions are difficult to construct, understand, maintain, and enhance.
  - GCC has become a hacker's paradise instead of a clean, production quality compiler generation framework.

Improving Retargetability and Instruction Selection in GCC

- Our Goals:
- Current Status:

Improving Retargetability and Instruction Selection in GCC

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
- Current Status:

Improving Retargetability and Instruction Selection in GCC

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:

Improving Retargetability and Instruction Selection in GCC

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:
  - A methodology of incremental construction has been devised.

Improving Retargetability and Instruction Selection in GCC

- Our Goals:
  - Discover the abstractions required in machine descriptions and develop a systematic methodology of constructing them.
  - Use tree tiling based instruction selection algorithms to allow for cleaner and simpler machine descriptions.
- Current Status:
  - A methodology of incremental construction has been devised.
  - Preliminary investigations in using `iburg` seem very promising. (Only 200 rules required for `i386` instead of over a 1000!)

Part 5

Improving Optimizations in GCC

Improving Machine Independent Optimizations in GCC

- The Problems:
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
    - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
    - Whole program analysis does not exist.
- Our Goals:
    - Implement scalable context and flow sensitive pointer analysis.
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:

Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - *gdfa*: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)

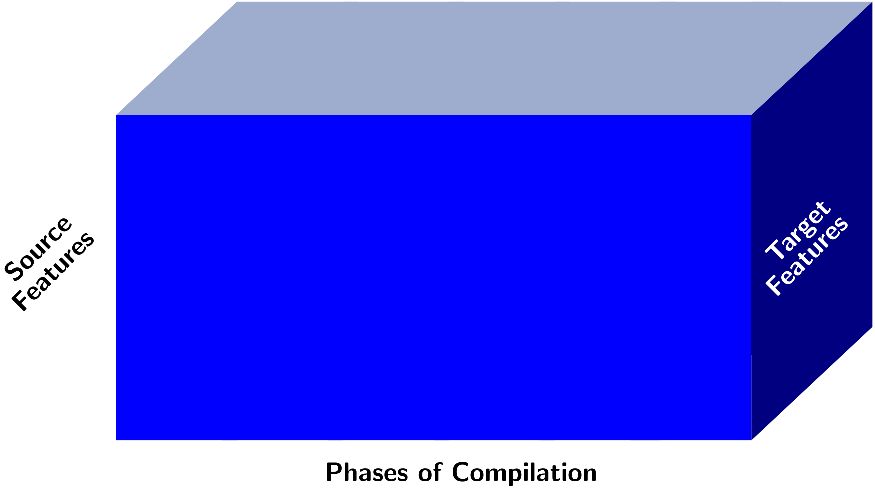
Improving Machine Independent Optimizations in GCC

- The Problems:
  - Primitive algorithms and adhoc designs (too many passes, repetitive work in passes, inappropriateness of IR).
  - Whole program analysis does not exist.
- Our Goals:
  - Implement scalable context and flow sensitive pointer analysis.
  - Facilitate generation of optimizers from specifications.
    - Clean specifications
    - Systematic local, global, and interprocedural analysis
    - Simple, efficient, generic, and precise algorithms
    - Incremental analyses for aggressive optimizations
- Current Status:
  - *gdfa*: Generic intraprocedural bit vector data flow analysis (patch released for GCC 4.3.0)
  - Algorithms and formal theory required further is in place.

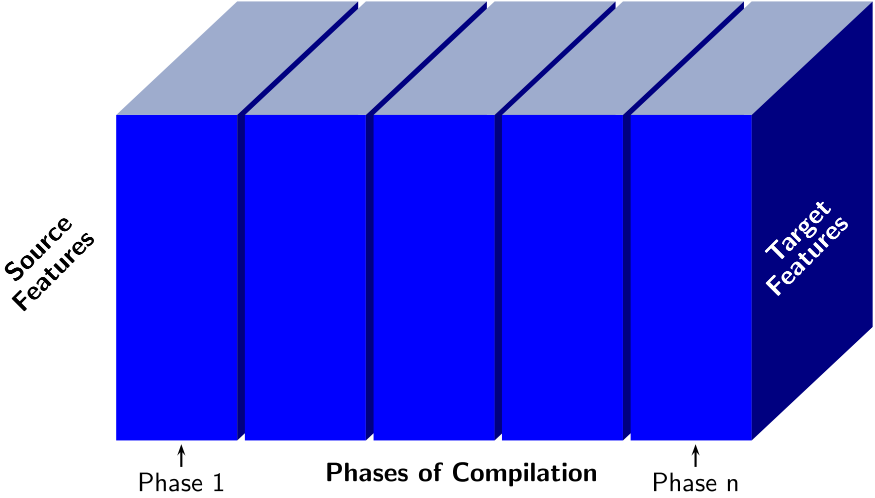
Part 6

Systematic Construction of Machine Descriptions

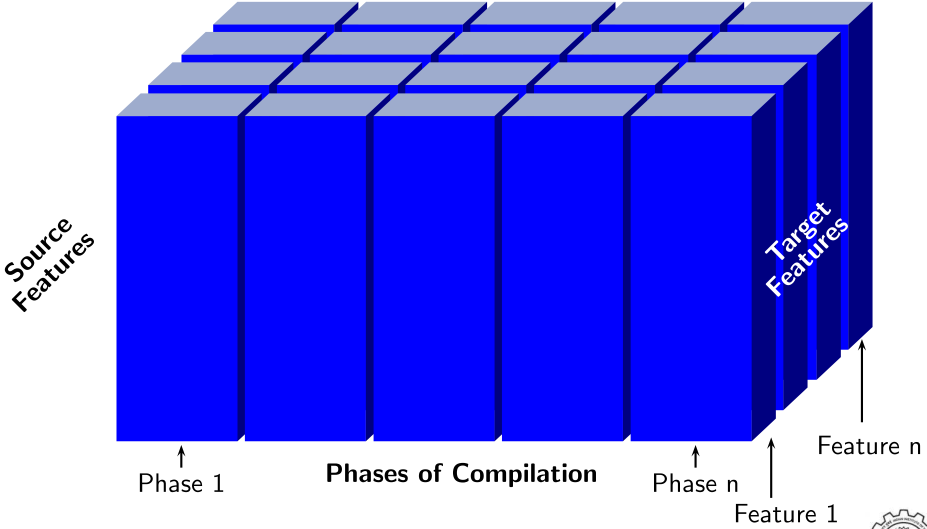
In Search of Modularity in Retargetable Compilation



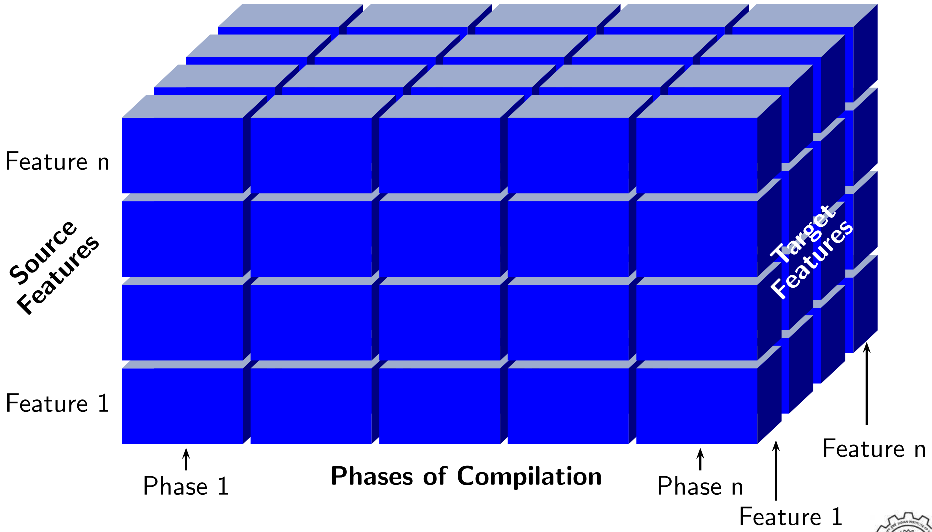
In Search of Modularity in Retargetable Compilation



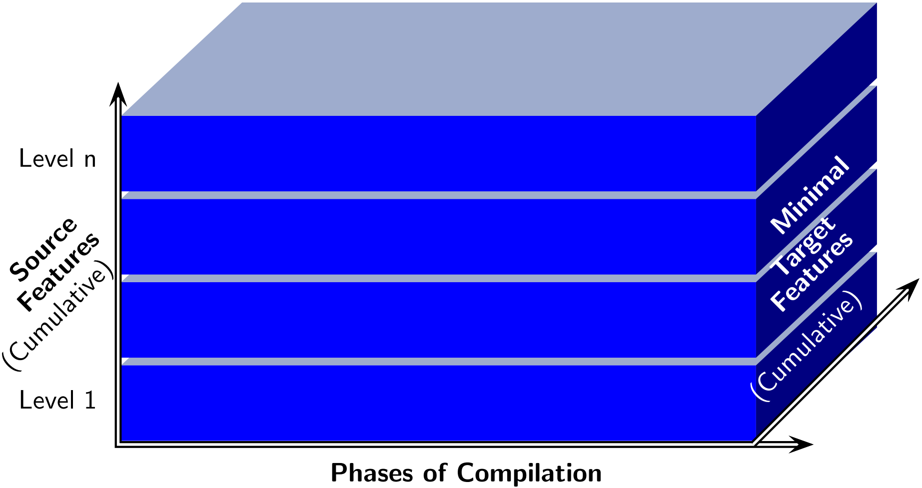
In Search of Modularity in Retargetable Compilation



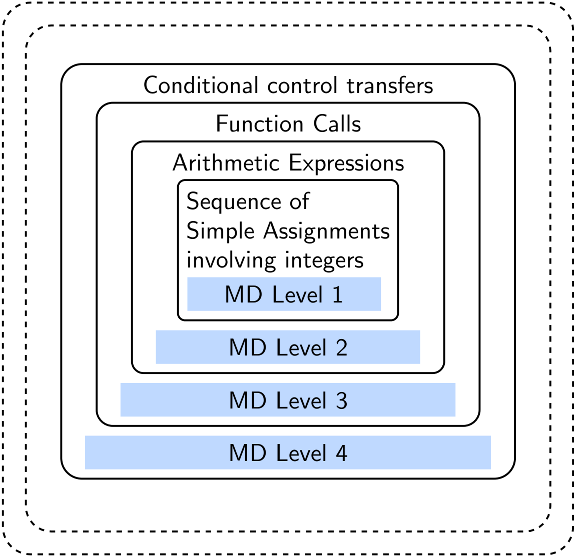
In Search of Modularity in Retargetable Compilation



In Search of Modularity in Retargetable Compilation



Systematic Development of Machine Descriptions [GREPS 2007]



Part 8

Translation Validation of GCC

Translation Validation of GCC

- Problem:
- Our Objectives:
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives:
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program.
  - Derive the conditions under which the observables correspond at the start of the program.
- Current Status:
- Future work:

Translation Validation of GCC

- Problem:
    - Establishing correctness of compilers is important.
    - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
    - Define suitable observation points and observables
    - Establish the conditions under which the observables correspond at the end of the program.
    - Derive the conditions under which the observables correspond at the start of the program.
- Current Status: Formal theory and prototype implementation to show the correctness of translation of a few programs exist.
- Future work:

Translation Validation of GCC

- Problem:
  - Establishing correctness of compilers is important.
  - Verifying a real compiler is very difficult.
- Our Objectives: To build a system to verify the correctness of the translation of given program.
- Our approach:
  - Define suitable observation points and observables
  - Establish the conditions under which the observables correspond at the end of the program.
  - Derive the conditions under which the observables correspond at the start of the program.
- Current Status: Formal theory and prototype implementation to show the correctness of translation of a few programs exist.
- Future work:
  - Cleaning up the theory to systematize the termination criteria.
  - Extending the approach to include more optimizations.

Part 9

Linear Types in GCC

Linear Types in GCC

- The Problems:

- Our Goals:

- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.

- Our Goals:

- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.
  - Variants of linearity have been identified.

Linear Types in GCC

- The Problems:
  - Aliases created by pointers is a major problem in C.
  - Significant imprecision in analysis
  - The scope of optimizations is significantly reduced.
  - Parallelization and Vectorization becomes difficult.
  - Synchronization and correctness problems in threads.
- Our Goals:
  - Use linear types to prohibit aliasing.
  - Allow reasonable limited relaxations of linearity constraints.
  - Define appropriate type system and enforce it.
- Current Status:
  - Linearity aspects in C have been studied in details.
  - Variants of linearity have been identified.
  - An initial draft of the type system is in place.

Part 10

Conclusions

Conclusions

- GCC Resource Center at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC

Conclusions

- GCC Resource Center at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC

Conclusions

- GCC Resource Center at IIT Bombay
  - Synergy from group activities
  - Long term commitment to challenging research problems
  - A desire to explore real issues in real compilers
    A dream to improve GCC
- *Would you like to be a part of this dream?*

Last but not the least . . .

Thank You!