Workshop on Essential Abstractions in GCC

An Overview of Compilation and GCC

GCC Resource Center

(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

29 June 2013

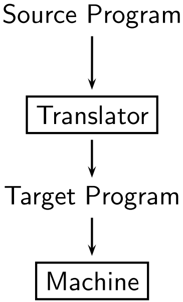
Outline

- Introduction to Compilation
- An Overview of Compilation Phases
- An Overview of GCC

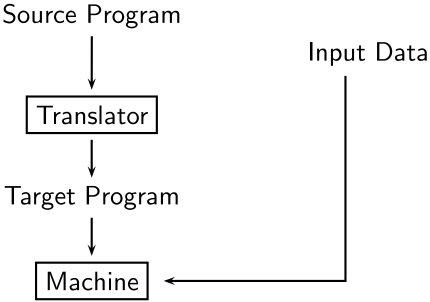
Part 1

Introduction to Compilation

Implementation Mechanisms



Implementation Mechanisms



Source Program

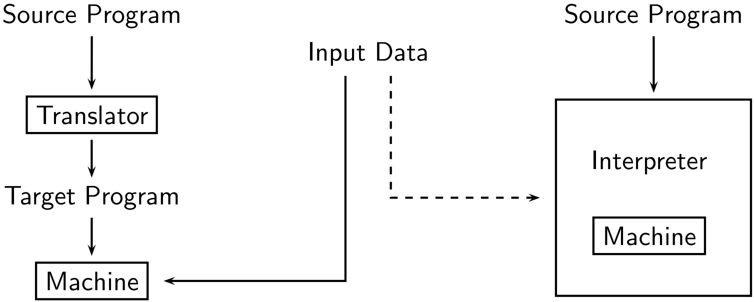
Translator

Target Program

Machine

Input Data

Implementation Mechanisms


Source Program

Translator

Target Program

Machine

Input Data

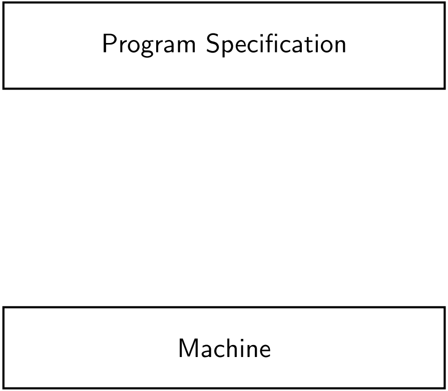
Source Program

Interpreter

Machine

Implementation Mechanisms as “Bridges”

- “Gap” between the “levels” of program specification and execution

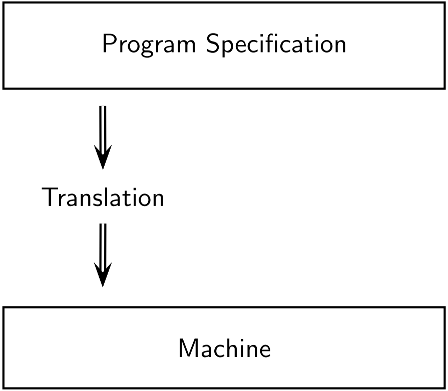


Program Specification

Machine

Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



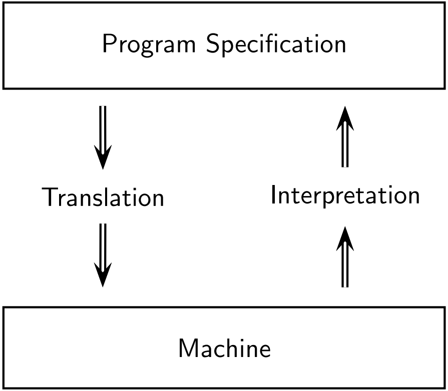
Program Specification

Translation

Machine

Implementation Mechanisms as "Bridges"

- "Gap" between the "levels" of program specification and execution



Program Specification

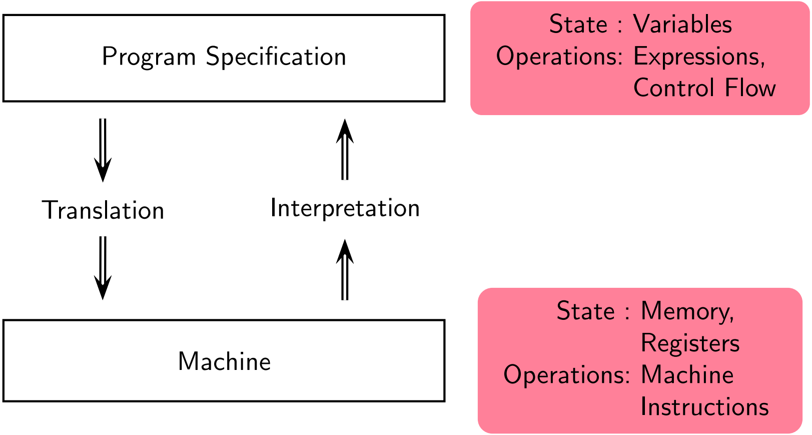
Translation

Interpretation

Machine

Implementation Mechanisms as “Bridges”

- “Gap” between the “levels” of program specification and execution



Program Specification

State : Variables
Operations: Expressions, Control Flow

Translation

Interpretation

Machine

State : Memory, Registers
Operations: Machine Instructions

High and Low Level Abstractions

Input C statement

```
a = b<10?b:c;
```

Spim Assembly Equivalent

```
     lw   $t0, 4($fp)   ;     t0 <- b          # Is b smaller
     slti $t0, $t0, 10  ;     t0 <- t0 < 10    # than 10?
     not  $t0, $t0      ;     t0 <- !t0
     bgtz $t0, L0:      ;     if t0>0 goto L0
     lw   $t0, 4($fp)   ;     t0 <- b          # YES
     b    L1:           ;     goto L1
 L0: lw   $t0, 8($fp)   ;L0: t0 <- c          # NO
 L1: sw   0($fp), $t0   ;L1: a <- t0
```

Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution

Implementation Mechanisms

- Translation = Analysis + Synthesis

  Interpretation = Analysis + Execution

- Translation   Instructions $\Longrightarrow$ Equivalent Instructions

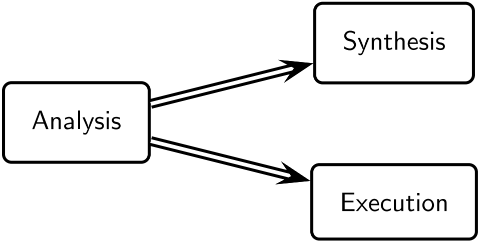
Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution

- Translation Instructions ⟹ Equivalent Instructions

  Interpretation Instructions ⟹ Actions Implied by Instructions

Language Implementation Models



Analysis

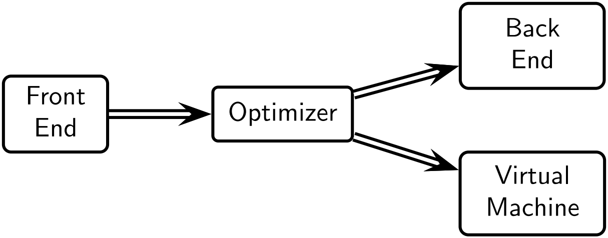
Synthesis

Execution

Compilation

Interpretation

Language Processor Models



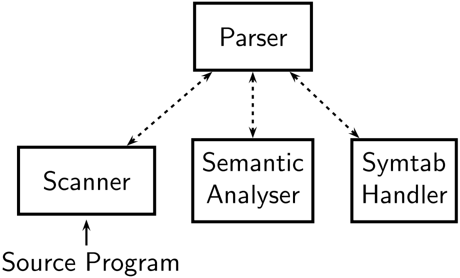
C,C++

Java, C#

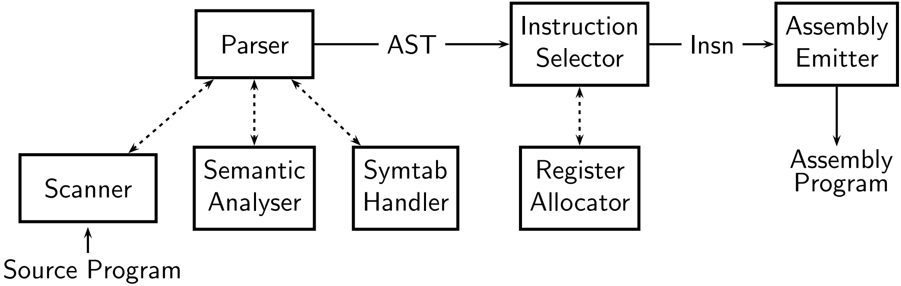
Part 2

An Overview of Compilation Phases

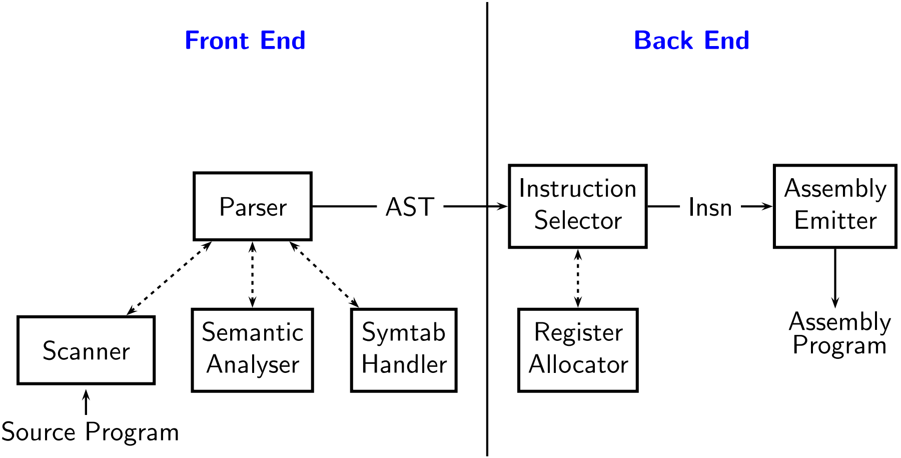
The Structure of a Simple Compiler



The Structure of a Simple Compiler



The Structure of a Simple Compiler

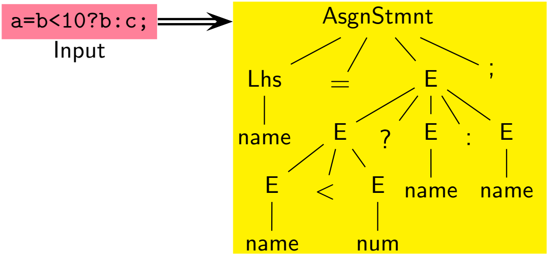


Translation Sequence in Our Compiler: Parsing

```
a=b<10?b:c;
```

Input

Translation Sequence in Our Compiler: Parsing



Parse Tree

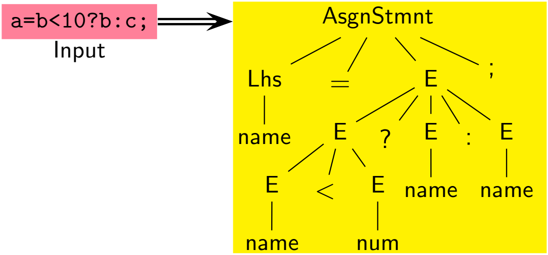
Issues:

- Grammar rules, terminals, non-terminals
- Order of application of grammar rules

  eg. is it `(a = b<10?)` followed by `(b:c)`?
- Values of terminal symbols

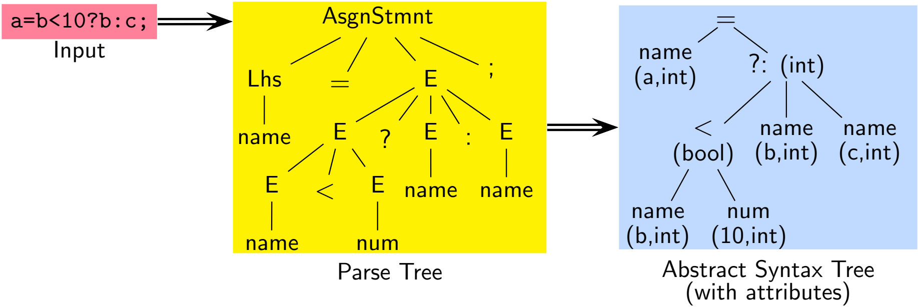
  eg. string "10" vs. integer number 10.

Translation Sequence in Our Compiler: Semantic Analysis



Parse Tree

Translation Sequence in Our Compiler: Semantic Analysis



Issues:

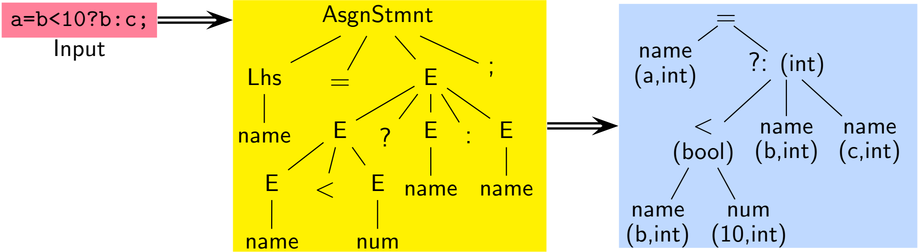
- Symbol tables

  Have variables been declared? What are their types? What is their scope?

- Type consistency of operators and operands

  The result of computing `b<10?` is bool and not int

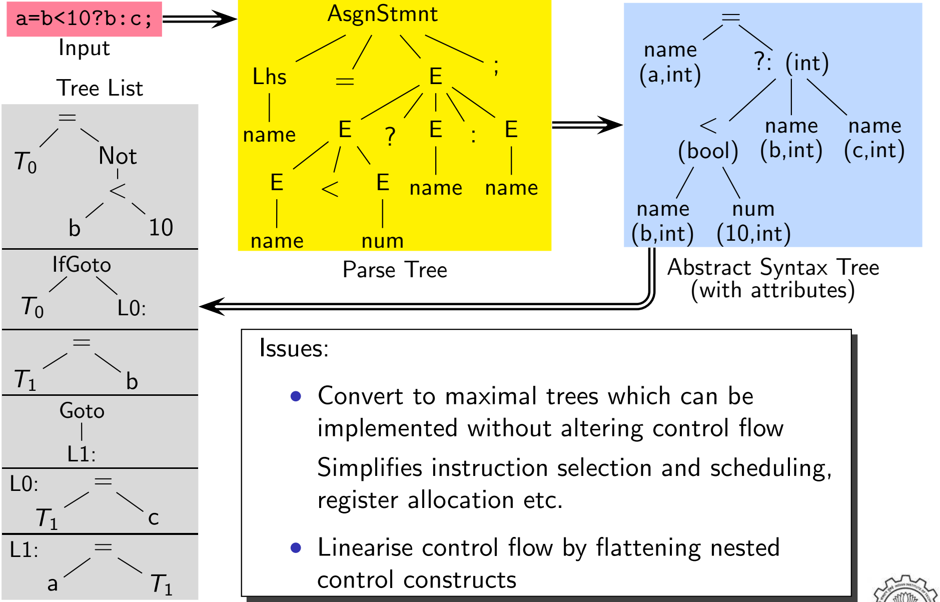
Translation Sequence in Our Compiler: IR Generation



Parse Tree

Abstract Syntax Tree
(with attributes)

Translation Sequence in Our Compiler: IR Generation



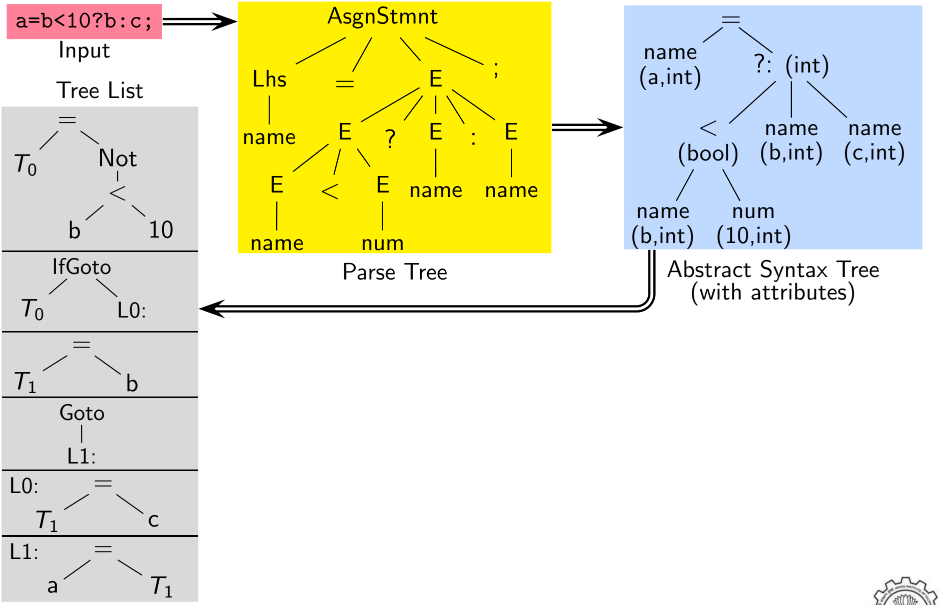
Issues:

- Convert to maximal trees which can be implemented without altering control flow

  Simplifies instruction selection and scheduling, register allocation etc.

- Linearise control flow by flattening nested control constructs

Translation Sequence in Our Compiler: Instruction Selection



`a=b<10?b:c;`

Input

AsgnStmnt
Lhs = E ;
name
E ? E : E
E < E name name
name num

Parse Tree

=
name (a,int)
?: (int)
< (bool)
name (b,int)
name (c,int)
name (b,int)
num (10,int)

Abstract Syntax Tree (with attributes)

Tree List

= T_0 Not < b 10

IfGoto T_0 L0:

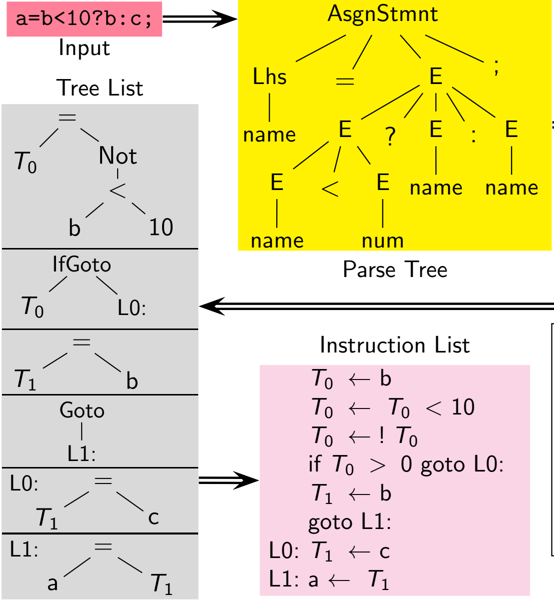
= T_1 b

Goto L1:

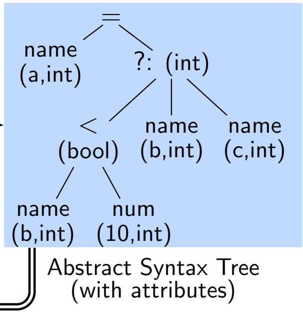
L0: = T_1 c

L1: = a T_1

Translation Sequence in Our Compiler: Instruction Selection



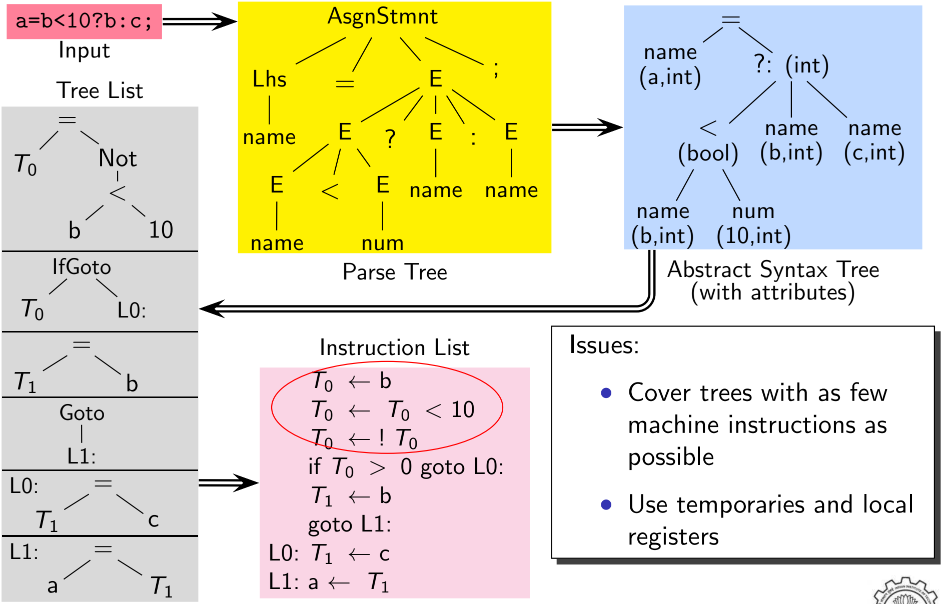
```
      T0 ← b
      T0 ← T0 < 10
      T0 ← ! T0
      if T0 > 0 goto L0:
      T1 ← b
      goto L1:
L0:  T1 ← c
L1:  a ← T1
```



Issues:

- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

Translation Sequence in Our Compiler: Instruction Selection



Instruction List

$T_0 \leftarrow$ b

$T_0 \leftarrow T_0 < 10$

$T_0 \leftarrow !\ T_0$

if $T_0 > 0$ goto L0:

$T_1 \leftarrow$ b

goto L1:

L0: $T_1 \leftarrow$ c

L1: a $\leftarrow T_1$

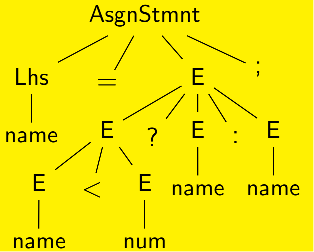
Issues:

- Cover trees with as few machine instructions as possible
- Use temporaries and local registers

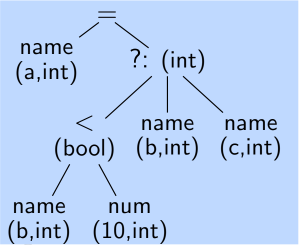
Translation Sequence in Our Compiler: Emitting Instructions

`a=b<10?b:c;`

Input

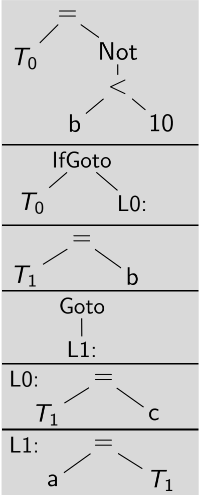


Parse Tree



Abstract Syntax Tree
(with attributes)

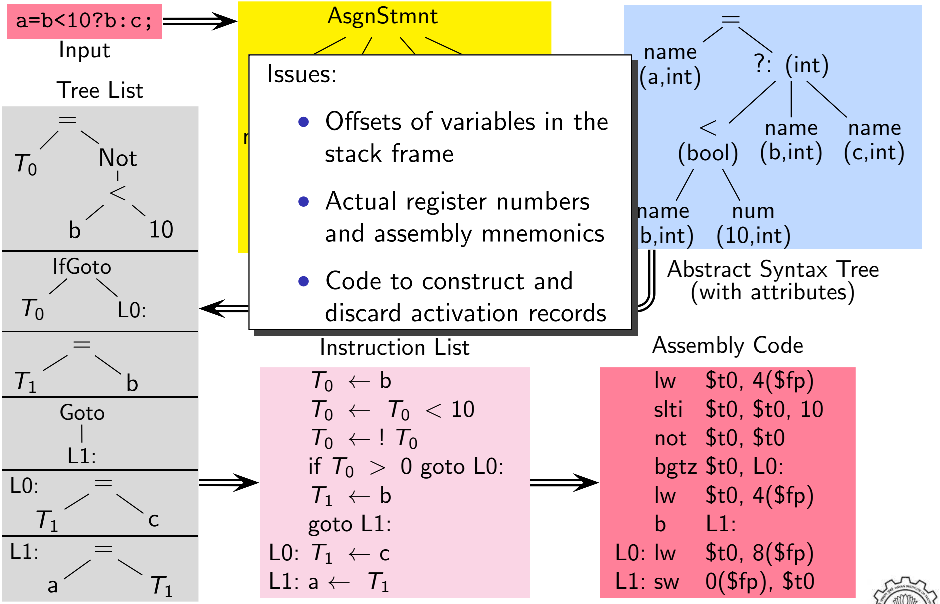
Tree List



Instruction List

$T_0 \leftarrow$ b
$T_0 \leftarrow T_0 < 10$
$T_0 \leftarrow ! \; T_0$
if $T_0 > 0$ goto L0:
$T_1 \leftarrow$ b
goto L1:
L0: $T_1 \leftarrow$ c
L1: a $\leftarrow T_1$

Translation Sequence in Our Compiler: Emitting Instructions



`a=b<10?b:c;`

Input

AsgnStmnt

Issues:

- Offsets of variables in the stack frame
- Actual register numbers and assembly mnemonics
- Code to construct and discard activation records

Abstract Syntax Tree (with attributes)

Tree List

Instruction List

$$
\begin{array}{ll}
 & T_0 \leftarrow \text{b} \\
 & T_0 \leftarrow T_0 < 10 \\
 & T_0 \leftarrow !\, T_0 \\
 & \text{if } T_0 > 0 \text{ goto L0:} \\
 & T_1 \leftarrow \text{b} \\
 & \text{goto L1:} \\
\text{L0:} & T_1 \leftarrow \text{c} \\
\text{L1:} & \text{a} \leftarrow T_1
\end{array}
$$

Assembly Code

```
    lw   $t0, 4($fp)
    slti $t0, $t0, 10
    not  $t0, $t0
    bgtz $t0, L0:
    lw   $t0, 4($fp)
    b    L1:
L0: lw   $t0, 8($fp)
L1: sw   0($fp), $t0
```

Part 3

Compilation Models

Compilation Models

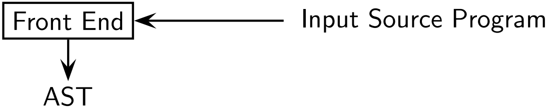
Aho Ullman Model

Davidson Fraser Model

Compilation Models

Aho Ullman Model

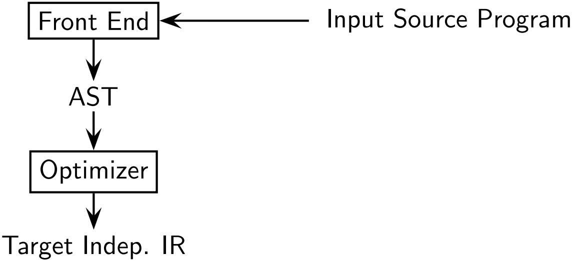
Davidson Fraser Model



Compilation Models

Aho Ullman Model

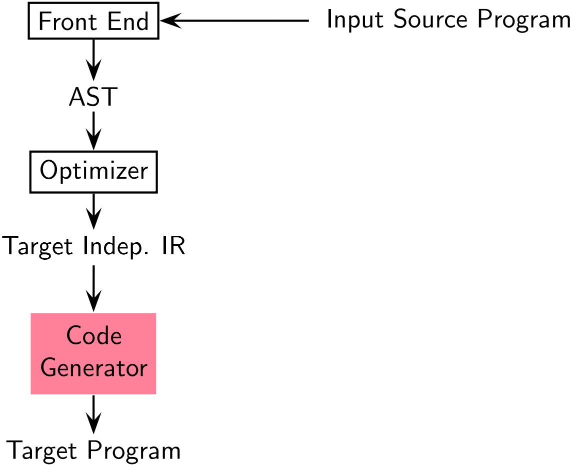
Davidson Fraser Model



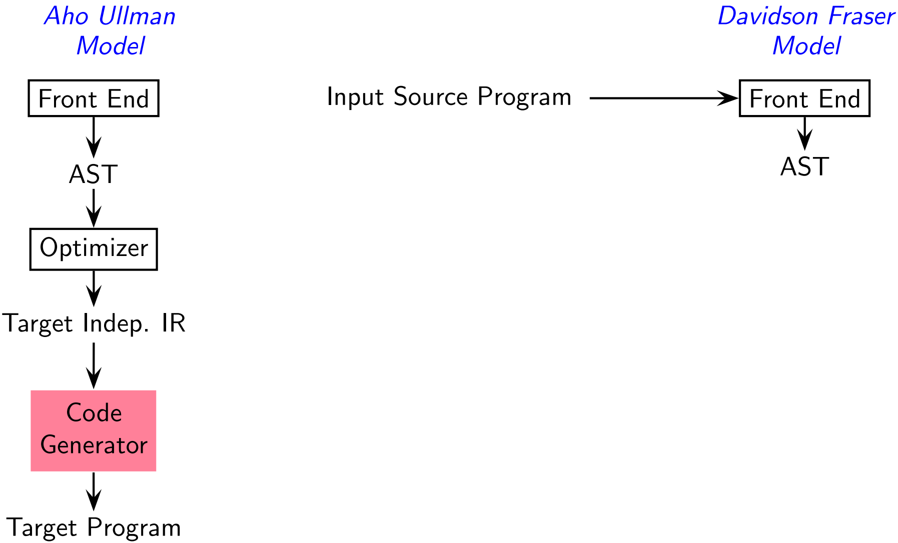
Compilation Models

Aho Ullman Model

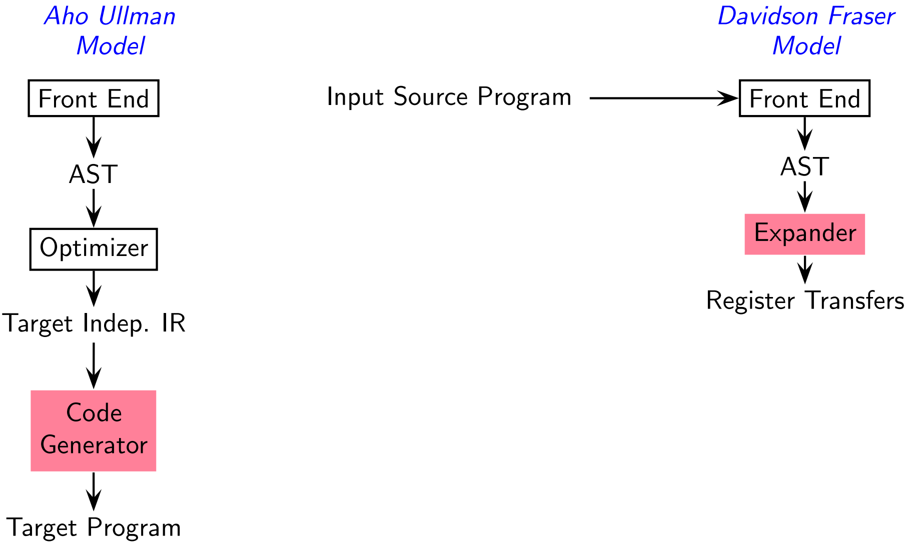
Davidson Fraser Model



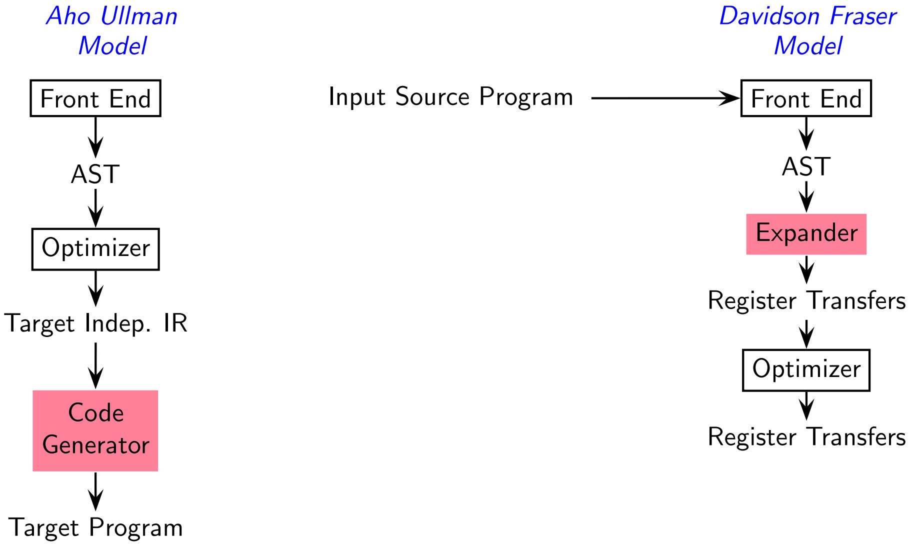
Compilation Models



Compilation Models



Compilation Models



Aho Ullman Model

Front End

AST

Optimizer

Target Indep. IR

Code Generator

Target Program

Input Source Program

Davidson Fraser Model

Front End

AST

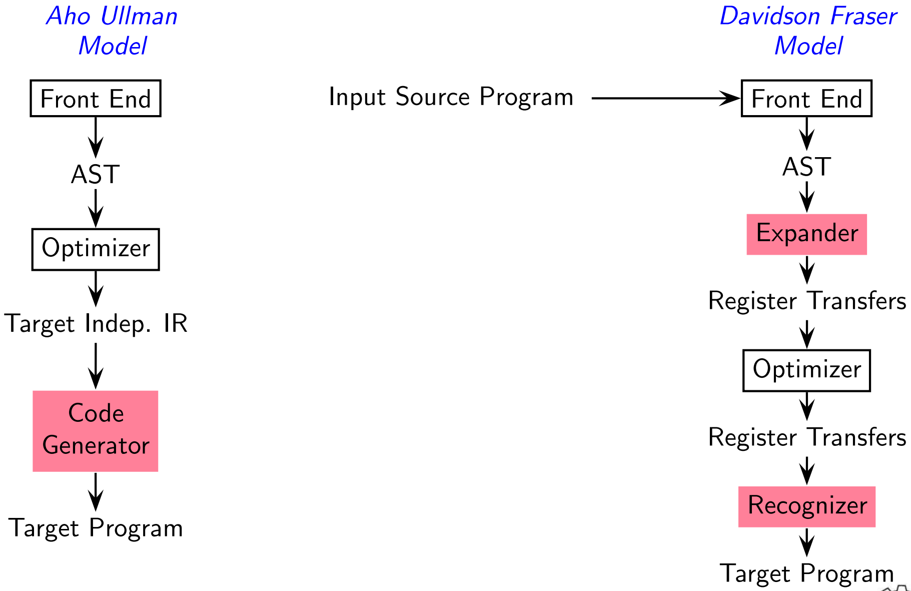
Expander

Register Transfers

Optimizer

Register Transfers

Compilation Models



Aho Ullman Model

Front End

AST

Optimizer

Target Indep. IR

Code Generator

Target Program

Input Source Program

Davidson Fraser Model

Front End

AST

Expander

Register Transfers

Optimizer

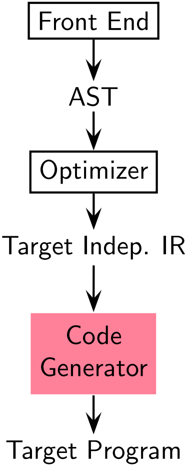
Register Transfers

Recognizer

Target Program

Compilation Models

Aho Ullman Model



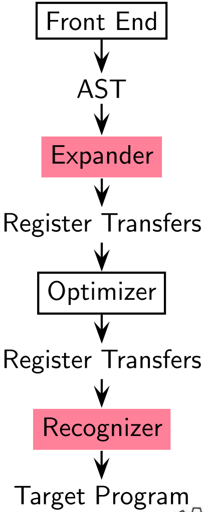
Aho Ullman: Instruction selection

- over optimized IR using
- cost based tree tiling matching

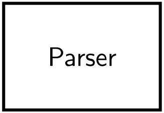
Davidson Fraser: Instruction selection

- over AST using
- simple full tree matching based algorithms that generate
- naive code which is
  - target dependent, and is
  - optimized subsequently

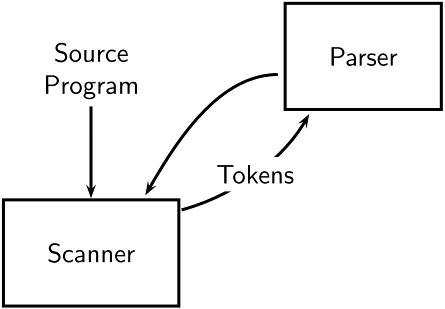
Davidson Fraser Model



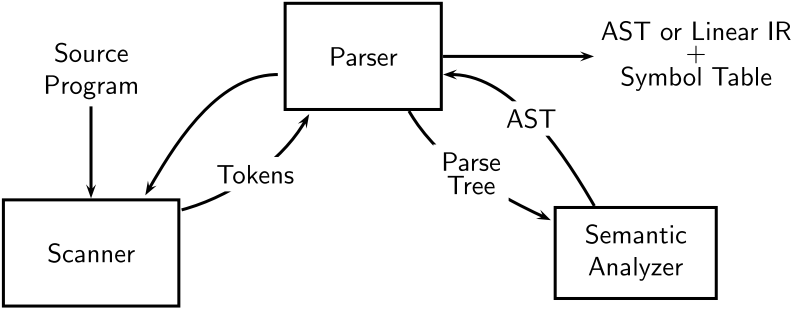
Typical Front Ends



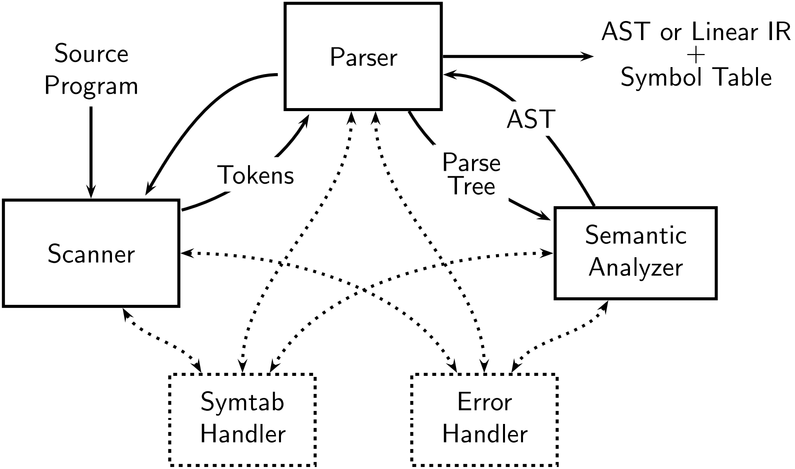
Typical Front Ends



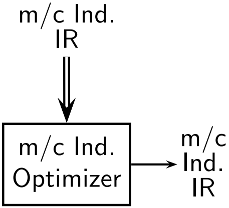
Typical Front Ends



Typical Front Ends

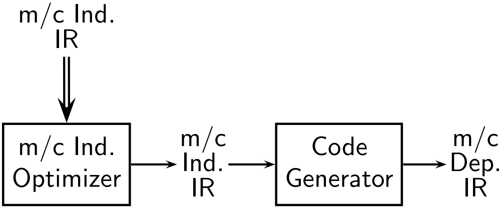


Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

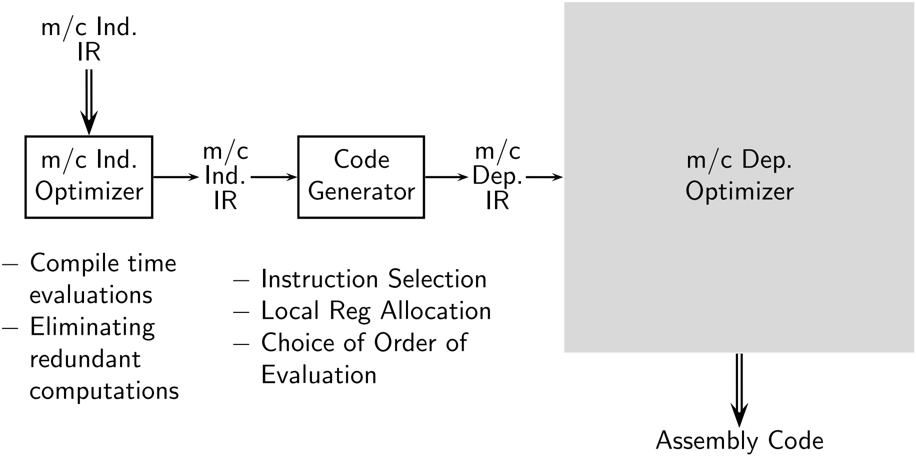
Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Typical Back Ends in Aho Ullman Model



m/c Ind. IR

m/c Ind. Optimizer

m/c Ind. IR

Code Generator

m/c Dep. IR

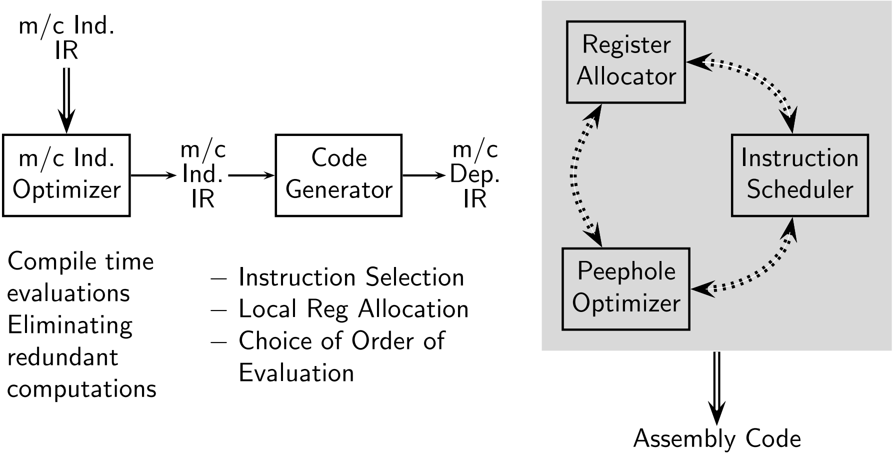
m/c Dep. Optimizer

Assembly Code

- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Typical Back Ends in Aho Ullman Model



- Compile time evaluations
- Eliminating redundant computations

- Instruction Selection
- Local Reg Allocation
- Choice of Order of Evaluation

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davisdon Fraser Model
Instruction Selection	• Machine independent IR is expressed in the form of trees • Machine instructions are described in the form of trees • Trees in the IR are “covered” using the instruction trees	
Optimization		

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td></td></tr>
<tr><td>Optimization</td><td></td><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davisdon Fraser Model
Instruction Selection	• Machine independent IR is expressed in the form of trees • Machine instructions are described in the form of trees • Trees in the IR are “covered” using the instruction trees	
	Cost based tree pattern matching	Structual tree pattern matching
Optimization		

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davisdon Fraser Model
Instruction Selection	• Machine independent IR is expressed in the form of trees • Machine instructions are described in the form of trees • Trees in the IR are "covered" using the instruction trees	
	Cost based tree pattern matching	Structual tree pattern matching
Optimization	Machine independent	

Retargetability in Aho Ullman and Davidson Fraser Models

	Aho Ullman Model	Davisdon Fraser Model
Instruction Selection	• Machine independent IR is expressed in the form of trees • Machine instructions are described in the form of trees • Trees in the IR are "covered" using the instruction trees	
	Cost based tree pattern matching	Structual tree pattern matching
Optimization	Machine independent	Machine dependent

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structual tree pattern matching</td></tr>
<tr><td rowspan="2">Optimization</td><td rowspan="2">Machine independent</td><td>Machine dependent</td></tr>
<tr><td>Key Insight: <i>Register transfers are target specific but their form is target independent</i></td></tr>
</table>

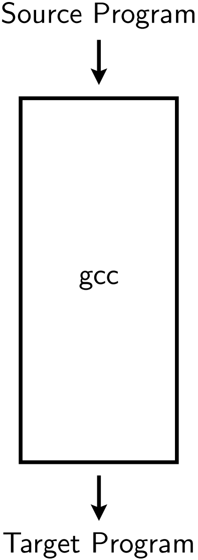
Part 4

GCC $\equiv$ The Great Compiler Challenge

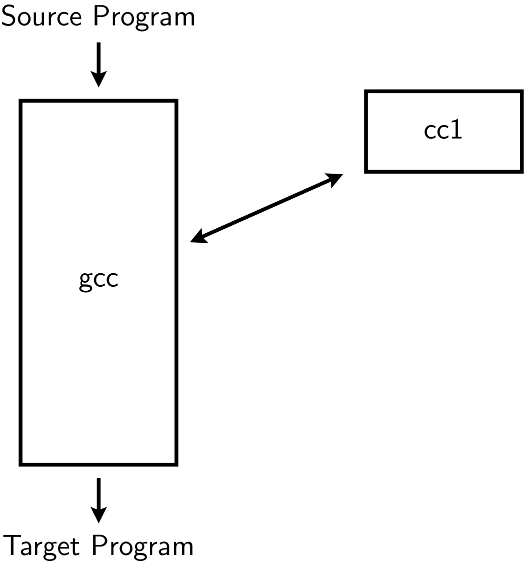
What is GCC?

- For the GCC developer community: The GNU Compiler Collection
- For other compiler writers: The Great Compiler Challenge ☺

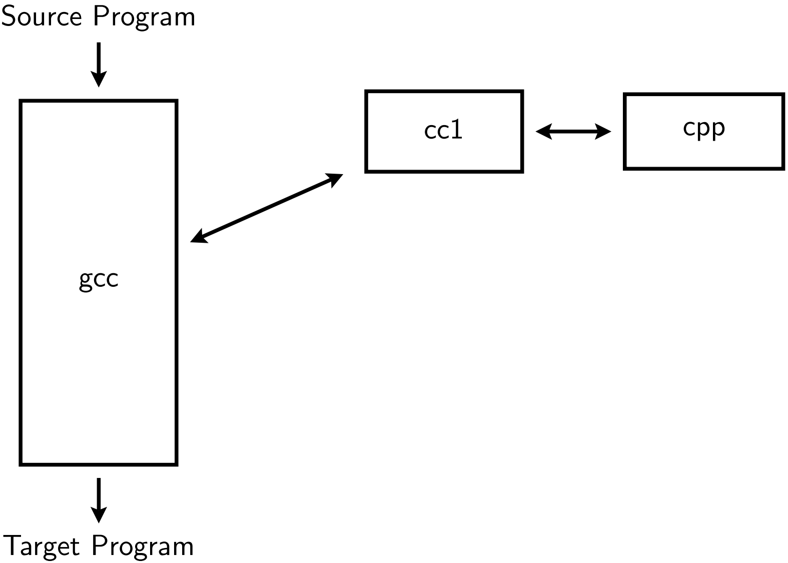
The GNU Tool Chain for C



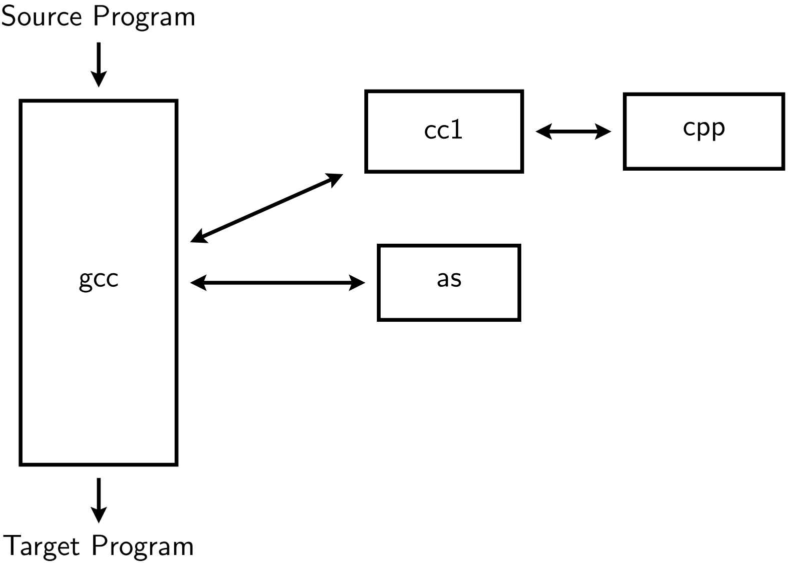
The GNU Tool Chain for C



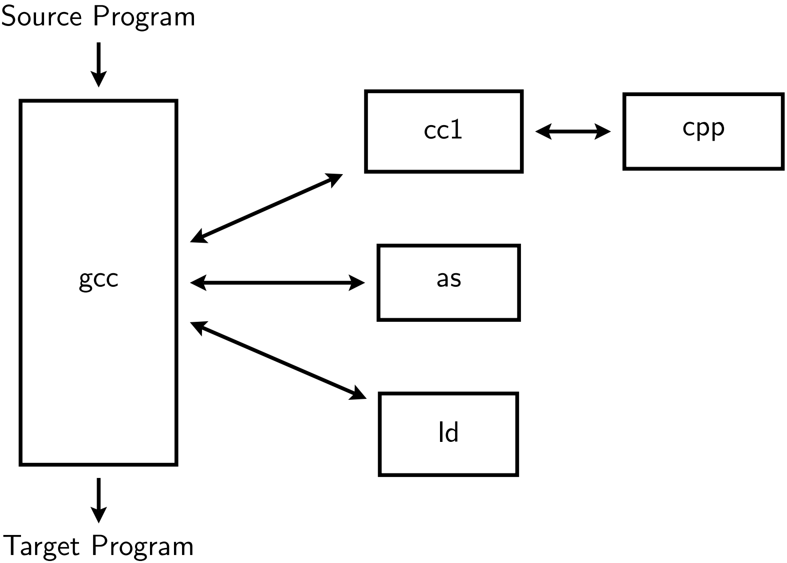
The GNU Tool Chain for C



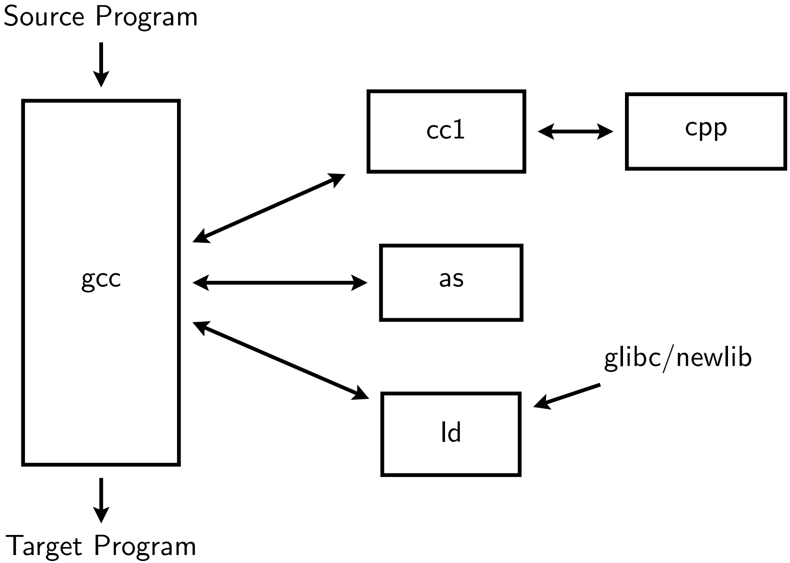
The GNU Tool Chain for C



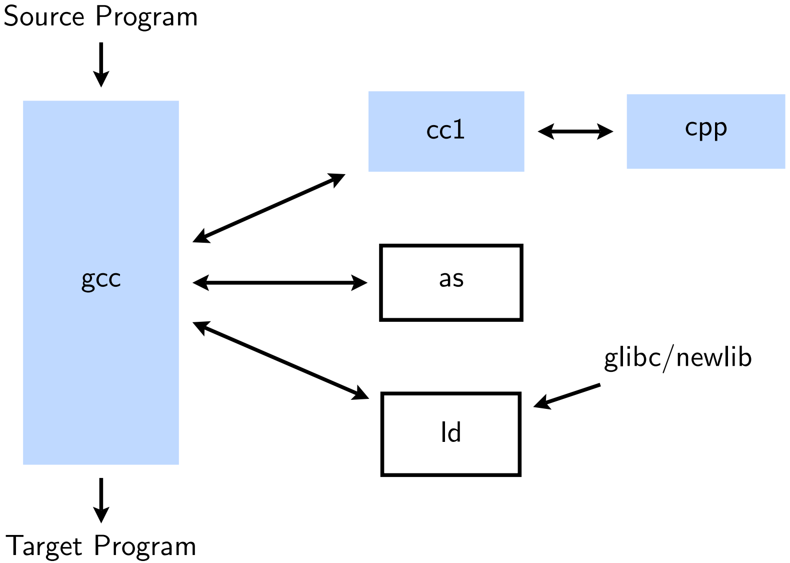
The GNU Tool Chain for C



The GNU Tool Chain for C



The GNU Tool Chain for C



Why is Understanding GCC Difficult?

Some of the obvious reasons:

- Comprehensiveness

  GCC is a production quality framework in terms of completeness and practical usefulness

- Open development model

  Could lead to heterogeneity. Design flaws may be difficult to correct

- Rapid versioning

  GCC maintenance is a race against time. Disruptive corrections are difficult

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

  Design, implement, test, release

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

  Design, implement, test, release

- Bazaar: Total Decentralization

  Release early, release often, make users partners in software development

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

  Design, implement, test, release

- Bazaar: Total Decentralization

  Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

  Design, implement, test, release

- Bazaar: Total Decentralization

  Release early, release often, make users partners in software development

“Given enough eyeballs, all bugs are shallow”

Code errors, logical errors, and architectural errors

Open Source and Free Software Development Model

The Cathedral and the Bazaar [Eric S Raymond, 1997]

- Cathedral: Total Centralized Control

  Design, implement, test, release

- Bazaar: Total Decentralization

  Release early, release often, make users partners in software development

"Given enough eyeballs, all bugs are shallow"

Code errors, logical errors, and architectural errors

A combination of the two seems more sensible

The Current Development Model of GCC

GCC follows a combination of the Cathedral and the Bazaar approaches

- GCC Steering Committee: Free Software Foundation has given charge
  - Major policy decisions
  - Handling Administrative and Political issues
- Release Managers:
  - Coordination of releases
- Maintainers:
  - Usually area/branch/module specific
  - Responsible for design and implementation
  - Take help of reviewers to evaluate submitted changes

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86),
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC,
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa,
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant),

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada

- Processors supported in standard releases:
  - Common processors:

    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:

    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:

    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:

  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC,

Comprehensiveness of GCC: Wide Applicability

- Input languages supported:
  C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada
- Processors supported in standard releases:
  - Common processors:
    Alpha, ARM, Atmel AVR, Blackfin, HC12, H8/300, IA-32 (x86), x86-64, IA-64, Motorola 68000, MIPS, PA-RISC, PDP-11, PowerPC, R8C/M16C/M32C, SPU, System/390/zSeries, SuperH, SPARC, VAX
  - Lesser-known target processors:
    A29K, ARC, ETRAX CRIS, D30V, DSP16xx, FR-30, FR-V, Intel i960, IP2000, M32R, 68HC11, MCORE, MMIX, MN10200, MN10300, Motorola 88000, NS32K, ROMP, Stormy16, V850, Xtensa, AVR32
  - Additional processors independently supported:
    D10V, LatticeMico32, MeP, Motorola 6809, MicroBlaze, MSP430, Nios II and Nios, PDP-10, TIGCC (m68k variant), Z8000, PIC24/dsPIC, NEC SX architecture

Comprehensiveness of GCC: Size

- Overall size

	Subdirectories	Files
gcc-4.4.2	3794	62301
gcc-4.6.0	4383	71096
gcc-4.7.2	4658	76287

- Core size (src/gcc)

	Subdirectories	Files
gcc-4.4.2	257	30163
gcc-4.6.0	336	36503
gcc-4.7.2	402	40193

- Machine Descriptions (src/gcc/config)

	Subdirectories	.c files	.h files	.md files
gcc-4.4.2	36	241	426	206
gcc-4.6.0	42	275	466	259
gcc-4.7.2	43	103	452	290

ohcount: Line Count of gcc-4.4.2

Language	Files	Code	Comment	Comment %	Blank	Total
c	15638	1840245	394682	17.7%	366815	2601742
cpp	19622	872775	190744	17.9%	189007	1252526
java	6342	681656	643045	48.5%	169465	1494166
ada	4206	638557	294881	31.6%	218000	1151438
autoconf	76	445046	393	0.1%	58831	504270
make	82	110064	3268	2.9%	13270	126602
html	480	103080	5658	5.2%	21438	130176
fortranfixed	2164	73366	1570	2.1%	9454	84390
assembler	183	42460	9607	18.5%	7084	59151
shell	137	39347	8832	18.3%	5485	53664
fortranfree	690	11852	2582	17.9%	1414	15848
objective_c	395	10562	1768	14.3%	2951	15281
automake	61	6014	853	12.4%	956	7823
perl	24	4111	1138	21.7%	732	5981
scheme	1	2775	153	5.2%	328	3256
ocaml	5	2482	538	17.8%	328	3348
python	6	1135	211	15.7%	220	1566
awk	9	1127	324	22.3%	193	1644
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	497	99	16.6%	30	626
tcl	1	392	113	22.4%	72	577
haskell	48	149	0	0.0%	16	165
emacslisp	1	59	21	26.2%	4	84
matlab	2	57	0	0.0%	7	64
Total	50312	4938881	1567750	24.1%	1071986	7578617

Language	Files	Code	Comment	Comment %	Blank	Total
c	15638	1840245	394682	17.7%	366815	2601742
cpp	19622	872775	190744	17.9%	189007	1252526
java	6342	681656	643045	48.5%	169465	1494166
ada	4206	638557	294881	31.6%	218000	1151438
autoconf	76	445046	393	0.1%	58831	504270
make	82	110064	3268	2.9%	13270	126602
html	480	103080	5658	5.2%	21438	130176
fortranfixed	2164	73366	1570	2.1%	9454	84390
assembler	183	42460	9607	18.5%	7084	59151
shell	137	39347	8832	18.3%	5485	53664
fortranfree	690	11852	2582	17.9%	1414	15848
objective_c	395	10562	1768	14.3%	2951	15281
automake	61	6014	853	12.4%	956	7823
perl	24	4111	1138	21.7%	732	5981
scheme	1	2775	153	5.2%	328	3256
ocaml	5	2482	538	17.8%	328	3348
python	6	1135	211	15.7%	220	1566
awk	9	1127	324	22.3%	193	1644
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	497	99	16.6%	30	626
tcl	1	392	113	22.4%	72	577
haskell	48	149	0	0.0%	16	165
emacslisp	1	59	21	26.2%	4	84
matlab	2	57	0	0.0%	7	64
Total	50312	4938881	1567750	24.1%	1071986	7578617

ohcount: Line Count of gcc-4.6.0

Language	Files	Code	Comment	Comment %	Blank	Total
c	18463	2100237	444333	17.5%	418292	2962862
cpp	22002	985076	229541	18.9%	214781	1429398
java	6342	681938	645505	48.6%	169046	1496489
ada	4605	680043	315956	31.7%	234467	1230466
autoconf	91	405461	509	0.1%	62914	468884
html	457	168355	5669	3.3%	38146	212170
make	98	121545	3659	2.9%	15618	140822
fortranfixed	2936	99413	1927	1.9%	13659	114999
shell	148	48032	10451	17.9%	6586	65069
assembler	208	46727	10227	18.0%	7853	64807
xml	75	36036	282	0.8%	3827	40145
objective_c	866	28014	5000	15.1%	8115	41129
fortranfree	821	13857	3147	18.5%	1695	18699
tex	2	11060	5776	34.3%	1433	18269
scheme	6	11023	1010	8.4%	1205	13238
automake	67	9440	1038	9.9%	1456	11934
perl	28	4445	1316	22.8%	837	6598
ocaml	6	2814	576	17.0%	378	3768
xslt	20	2805	436	13.5%	563	3804
awk	11	1740	396	18.5%	257	2393
python	10	1725	322	15.7%	383	2430
css	24	1589	143	8.3%	332	2064
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	402	84	17.3%	13	499
tcl	1	392	113	22.4%	72	577
javascript	4	341	87	20.3%	35	463
haskell	49	153	0	0.0%	17	170
bat	3	7	0	0.0%	0	7
matlab	1	5	0	0.0%	0	5
Total	57359	5464598	1688150	23.6%	1202428	8355176

Language	Files	Code	Comment	Comment %	Blank	Total
c	18463	2100237	444333	17.5%	418292	2962862
cpp	22002	985076	229541	18.9%	214781	1429398
java	6342	681938	645505	48.6%	169046	1496489
ada	4605	680043	315956	31.7%	234467	1230466
autoconf	91	405461	509	0.1%	62914	468884
html	457	168355	5669	3.3%	38146	212170
make	98	121545	3659	2.9%	15618	140822
fortranfixed	2936	99413	1927	1.9%	13659	114999
shell	148	48032	10451	17.9%	6586	65069
assembler	208	46727	10227	18.0%	7853	64807
xml	75	36036	282	0.8%	3827	40145
objective_c	866	28014	5000	15.1%	8115	41129
fortranfree	821	13857	3147	18.5%	1695	18699
tex	2	11060	5776	34.3%	1433	18269
scheme	6	11023	1010	8.4%	1205	13238
automake	67	9440	1038	9.9%	1456	11934
perl	28	4445	1316	22.8%	837	6598
ocaml	6	2814	576	17.0%	378	3768
xslt	20	2805	436	13.5%	563	3804
awk	11	1740	396	18.5%	257	2393
python	10	1725	322	15.7%	383	2430
css	24	1589	143	8.3%	332	2064
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	402	84	17.3%	13	499
tcl	1	392	113	22.4%	72	577
javascript	4	341	87	20.3%	35	463
haskell	49	153	0	0.0%	17	170
bat	3	7	0	0.0%	0	7
matlab	1	5	0	0.0%	0	5
Total	57359	5464598	1688150	23.6%	1202428	8355176

`ohcount`: Line Count of gcc-4.7.2

Language	Files	Code	Comment	Comment %	Blank	Total
c	20857	2289353	472640	17.1%	449939	3211932
cpp	23370	1030227	243717	19.1%	224079	1498023
ada	4913	726638	334360	31.5%	252044	1313042
java	6342	681938	645506	48.6%	169046	1496490
autoconf	94	428267	523	0.1%	66647	495437
html	336	151194	5667	3.6%	33877	190738
fortranfixed	3256	112286	2010	1.8%	15599	129895
make	106	110762	3875	3.4%	13811	128448
xml	76	50179	571	1.1%	6048	56798
assembler	240	49903	10975	18.0%	8584	69462
shell	157	49148	10848	18.1%	6757	66753
objective_c	882	28226	5267	15.7%	8324	41817
fortranfree	872	14474	3445	19.2%	1817	19736
tex	2	11060	5776	34.3%	1433	18269
scheme	6	11023	1010	8.4%	1205	13238
automake	72	10496	1179	10.1%	1582	13257
perl	29	4551	1322	22.5%	854	6727
ocaml	6	2830	576	16.9%	378	3784
xslt	20	2805	436	13.5%	563	3804
awk	16	2103	556	20.9%	352	3011
python	10	1672	400	19.3%	400	2472
css	25	1590	143	8.3%	332	2065
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	402	84	17.3%	13	499
tcl	1	392	113	22.4%	72	577
javascript	3	208	87	29.5%	33	328
haskell	49	153	0	0.0%	17	170
matlab	2	57	0	0.0%	8	65
bat	3	7	0	0.0%	0	7
Total	61760	5773867	1751733	23.3%	1264262	8789862

Language	Files	Code	Comment	Comment %	Blank	Total
c	20857	2289353	472640	17.1%	449939	3211932
cpp	23370	1030227	243717	19.1%	224079	1498023
ada	4913	726638	334360	31.5%	252044	1313042
java	6342	681938	645506	48.6%	169046	1496490
autoconf	94	428267	523	0.1%	66647	495437
html	336	151194	5667	3.6%	33877	190738
fortranfixed	3256	112286	2010	1.8%	15599	129895
make	106	110762	3875	3.4%	13811	128448
xml	76	50179	571	1.1%	6048	56798
assembler	240	49903	10975	18.0%	8584	69462
shell	157	49148	10848	18.1%	6757	66753
objective_c	882	28226	5267	15.7%	8324	41817
fortranfree	872	14474	3445	19.2%	1817	19736
tex	2	11060	5776	34.3%	1433	18269
scheme	6	11023	1010	8.4%	1205	13238
automake	72	10496	1179	10.1%	1582	13257
perl	29	4551	1322	22.5%	854	6727
ocaml	6	2830	576	16.9%	378	3784
xslt	20	2805	436	13.5%	563	3804
awk	16	2103	556	20.9%	352	3011
python	10	1672	400	19.3%	400	2472
css	25	1590	143	8.3%	332	2065
pascal	4	1044	141	11.9%	218	1403
csharp	9	879	506	36.5%	230	1615
dcl	2	402	84	17.3%	13	499
tcl	1	392	113	22.4%	72	577
javascript	3	208	87	29.5%	33	328
haskell	49	153	0	0.0%	17	170
matlab	2	57	0	0.0%	8	65
bat	3	7	0	0.0%	0	7
Total	61760	5773867	1751733	23.3%	1264262	8789862

ohcount: Line Count of gcc-4.4.2/gcc

Language	Files	Code	Comment	Comment %	Blank	Total
c	13296	1254253	282582	18.4%	283766	1820601
ada	4196	636876	294321	31.6%	217401	1148598
cpp	7418	184186	52163	22.1%	54048	290397
fortranfixed	2086	67988	1521	2.2%	9079	78588
assembler	132	31092	7243	18.9%	4770	43105
autoconf	3	26996	10	0.0%	3383	30389
fortranfree	652	10898	2376	17.9%	1314	14588
objective_c	391	10155	1654	14.0%	2830	14639
make	3	5340	1027	16.1%	814	7181
scheme	1	2775	153	5.2%	328	3256
ocaml	5	2482	538	17.8%	328	3348
shell	16	2256	712	24.0%	374	3342
awk	7	1022	251	19.7%	187	1460
perl	1	772	205	21.0%	137	1114
haskell	48	149	0	0.0%	16	165
matlab	2	57	0	0.0%	7	64
Total	28258	2242738	647591	22.4%	579484	3469813

ohcount: Line Count of gcc-4.6.0/gcc

Language	Files	Code	Comment	Comment %	Blank	Total
c	15787	1462494	321820	18.0%	324179	2108493
ada	4595	678362	315396	31.7%	233868	1227626
cpp	8666	252213	61026	19.5%	67144	380383
fortranfixed	2850	93549	1878	2.0%	13260	108687
assembler	137	31548	7446	19.1%	4857	43851
autoconf	3	28775	12	0.0%	4020	32807
objective_c	861	27465	4822	14.9%	7967	40254
fortranfree	783	12903	2936	18.5%	1595	17434
scheme	6	11023	1010	8.4%	1205	13238
make	4	6078	1070	15.0%	893	8041
tex	1	5441	2835	34.3%	702	8978
ocaml	6	2814	576	17.0%	378	3768
shell	16	1980	597	23.2%	338	2915
awk	9	1635	323	16.5%	251	2209
perl	3	866	225	20.6%	158	1249
haskell	49	153	0	0.0%	17	170
matlab	1	5	0	0.0%	0	5
Total	33777	2617304	721972	21.6%	660832	4000108

`ohcount`: Line Count of gcc-4.7.2/gcc

Language	Files	Code	Comment	Comment %	Blank	Total
c	17849	1601863	335879	17.3%	344693	2282435
ada	4903	724957	333800	31.5%	251445	1310202
cpp	9563	275971	63875	18.8%	71647	411493
fortranfixed	3158	105987	1961	1.8%	15175	123123
autoconf	3	30014	12	0.0%	4139	34165
objective_c	877	28017	5109	15.4%	8249	41375
fortranfree	834	13516	3234	19.3%	1716	18466
scheme	6	11023	1010	8.4%	1205	13238
make	6	6248	1113	15.1%	916	8277
tex	1	5441	2835	34.3%	702	8978
ocaml	6	2830	576	16.9%	378	3784
shell	22	2265	735	24.5%	391	3391
awk	11	1646	390	19.2%	271	2307
perl	3	913	226	19.8%	163	1302
assembler	7	343	136	28.4%	27	506
haskell	49	153	0	0.0%	17	170
matlab	2	57	0	0.0%	8	65
Total	37300	2811244	750891	21.1%	701142	4263277

Why is Understanding GCC Difficult?

Deeper technical reasons

- GCC is not a compiler but a *compiler generation framework*

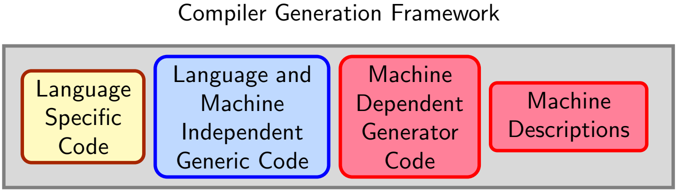
  Two distinct gaps that need to be bridged

  - Input-output of the generation framework
    The target specification and the generated compiler
  - Input-output of the generated compiler
    A source program and the generated assembly program

- GCC generated compiler uses a derivative of the Davidson-Fraser model of compilation

  - Early instruction selection
  - Machine dependent intermediate representation
  - Simplistic instruction selection and retargatibility mechanism

The Architecture of GCC



Compiler Generation Framework

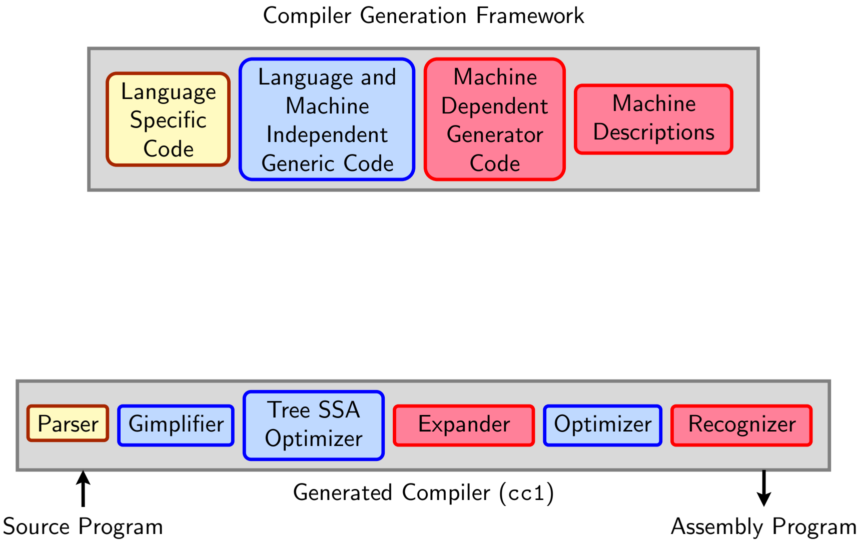
Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

The Architecture of GCC



Compiler Generation Framework

Language Specific Code

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Parser

Gimplifier

Tree SSA Optimizer

Expander

Optimizer

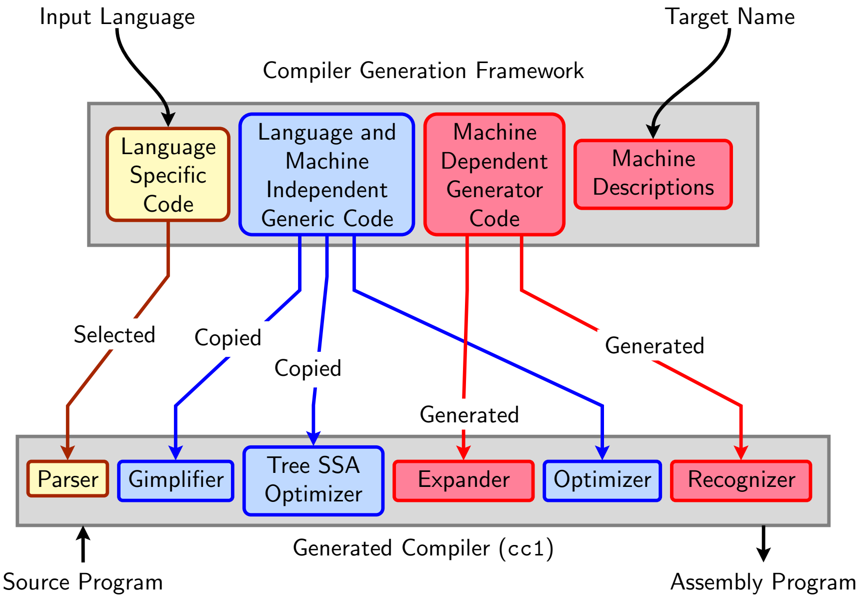
Recognizer

Generated Compiler (cc1)

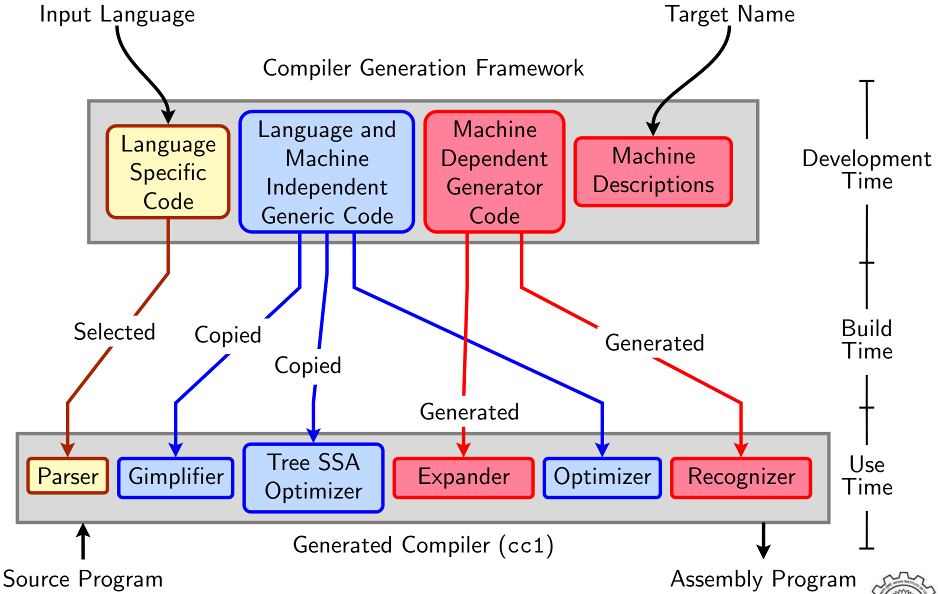
Source Program

Assembly Program

The Architecture of GCC



The Architecture of GCC



An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!

An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```

An Example of The Generation Related Gap

- Predicate function for invoking the loop distribution pass

```
static bool
gate_tree_loop_distribution (void)
{
  return flag_tree_loop_distribution != 0;
}
```

- There is no declaration of or assignment to variable `flag_tree_loop_distribution` in the entire source!
- It is described in `common.opt` as follows

```
ftree-loop-distribution
Common Report Var(flag_tree_loop_distribution) Optimization
Enable loop distribution on trees
```

- The required C statements are generated during the build

Another Example of The Generation Related Gap

- Locating the `main` function in `gcc-4.7.2/gcc` using cscope -R

Another Example of The Generation Related Gap

- Locating the `main` function in `gcc-4.7.2/gcc` using cscope -R

  8125 occurrences!

Another Example of The Generation Related Gap

- Locating the `main` function in `gcc-4.7.2/gcc` using cscope -R

  8125 occurrences!

- Number of main functions in the entire tarball

Another Example of The Generation Related Gap

- Locating the `main` function in `gcc-4.7.2/gcc` using cscope -R

  8125 occurrences!

- Number of main functions in the entire tarball

  12799!

Another Example of The Generation Related Gap

- Locating the `main` function in `gcc-4.7.2/gcc` using cscope -R

  8125 occurrences!

- Number of main functions in the entire tarball

  12799!

- What if we do not search recursively?

Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.7.2/gcc` using cscope

Another Example of The Generation Related Gap

Locating the main function in the directory gcc-4.7.2/gcc using cscope

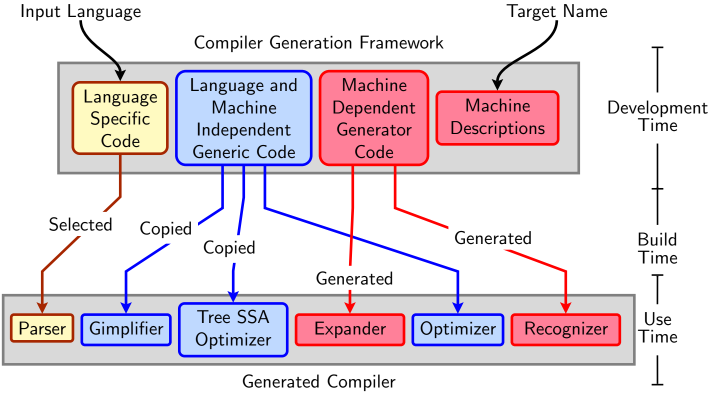
```
  File                    Line
0 s-oscons-tmplt.c         238 main (void ) {
1 collect2.c              1021 main (int argc, char **argv)
2 divtab-sh4-300.c          31 main ()
3 divtab-sh4.c              30 main ()
4 divtab.c                 131 main ()
5 gen-mul-tables.cc       1224 main ()
6 vms-ar.c                 122 main (int argc, char *argv[])
7 vms-ld.c                 559 main (int argc, char **argv)
8 fp-test.c                 85 main (void )
9 gcc-ar.c                  36 main(int ac, char **av)
a gcc.c                   6105 main (int argc, char **argv)
b gcov-dump.c               78 main (int argc ATTRIBUTE_UNUSED, cha
c gcov-iov.c                29 main (int argc, char **argv)
d gcov.c                   397 main (int argc, char **argv)
e genattr-common.c          64 main (int argc, char **argv)
f genattr.c                141 main (int argc, char **argv)
```

Another Example of The Generation Related Gap

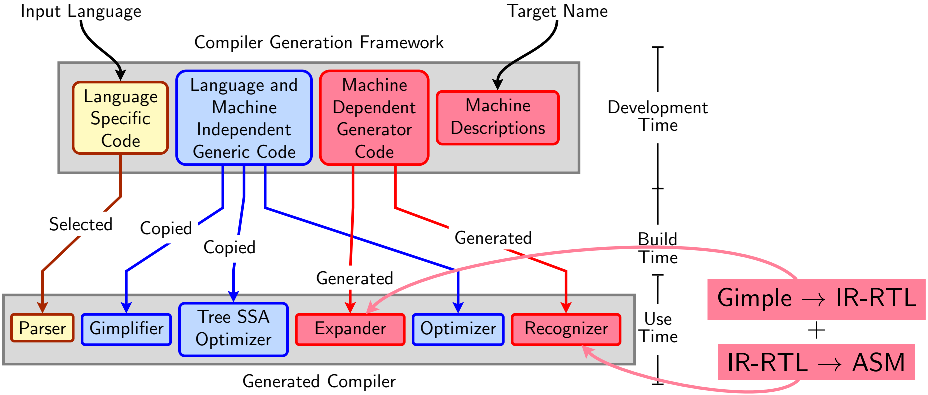
Locating the `main` function in the directory `gcc-4.7.2/gcc` using cscope

```
g genattrtab.c          4880 main (int argc, char **argv)
h genautomata.c         9617 main (int argc, char **argv)
i genchecksum.c           97 main (int argc, char ** argv)
j gencodes.c              51 main (int argc, char **argv)
k genconditions.c        209 main (int argc, char **argv)
l genconfig.c            261 main (int argc, char **argv)
m genconstants.c          79 main (int argc, char **argv)
n genemit.c              775 main (int argc, char **argv)
o genenums.c              48 main (int argc, char **argv)
p genextract.c           402 main (int argc, char **argv)
q genflags.c             251 main (int argc, char **argv)
r gengenrtl.c            286 main (void )
s gengtype.c            4925 main (int argc, char **argv)
t genhooks.c             342 main (int argc, char **argv)
u genmddeps.c             43 main (int argc, char **argv)
v genmodes.c            1388 main (int argc, char **argv)
w genopinit.c            504 main (int argc, char **argv)
x genoutput.c            997 main (int argc, char **argv)
```

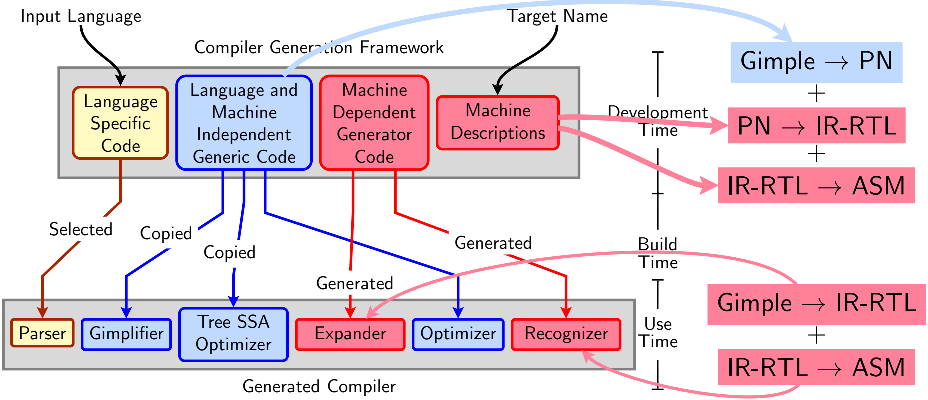
GCC Retargetability Mechanism



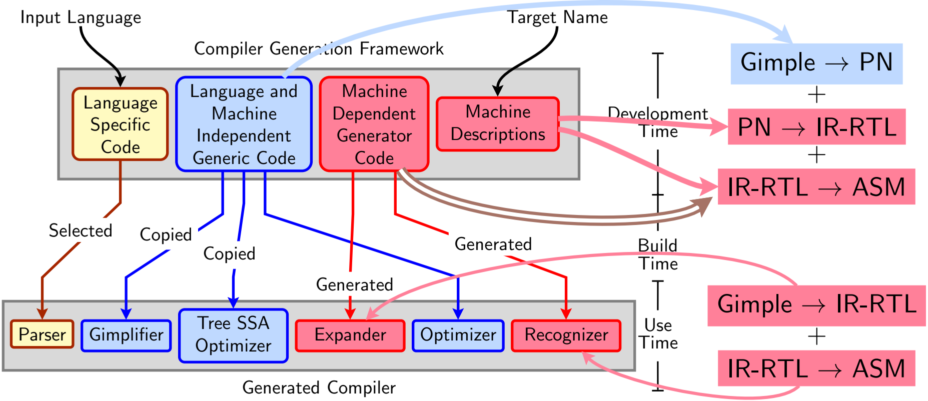
GCC Retargetability Mechanism



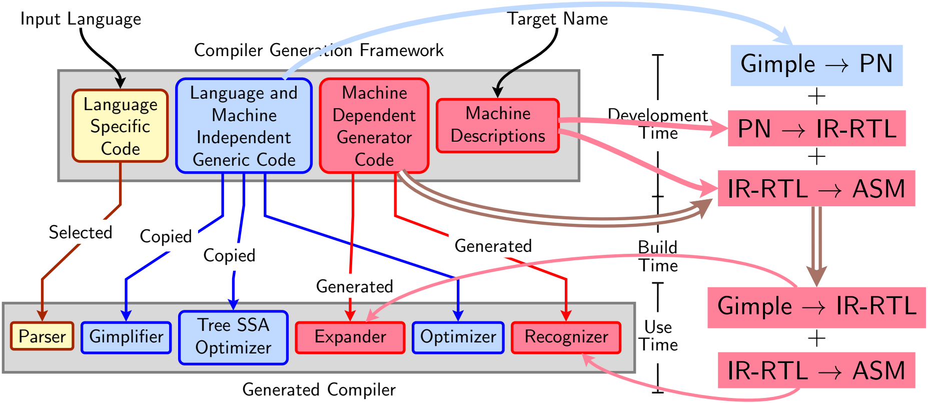
GCC Retargetability Mechanism



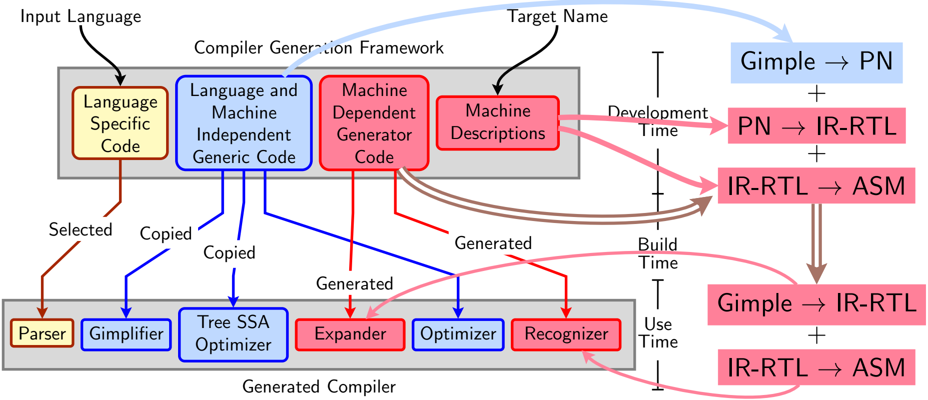
GCC Retargetability Mechanism



GCC Retargetability Mechanism



GCC Retargetability Mechanism



The generated compiler uses an adaptation of the Davison Fraser model

- Generic expander and recognizer
- Machine specific information is isolated in data structures
- Generating a compiler involves generating these data structures

The GCC Challenge: Poor Retargetability Mechanism

Symptoms:

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code

  Size in terms of line counts in `gcc-4.7.2` (counted using `wc -l`)

Files	i386	mips	arm
*.md	39582	16347	32385
*.c	41985	17761	26006
*.h	19174	5586	18012
Total	100741	39694	76403

The GCC Challenge: Poor Retargetability Mechanism

Symptoms:

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code

  Size in terms of line counts in `gcc-4.7.2` (counted using `wc -l`)

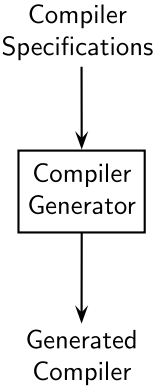
Files	i386	mips	arm
*.md	39582	16347	32385
*.c	41985	17761	26006
*.h	19174	5586	18012
Total	100741	39694	76403

- Machine descriptions are difficult to construct, understand, debug, and enhance

Meeting the GCC Challenge

Goal of Understanding	Methodology	Needs Examining		
		Makefiles	Source	MD
Translation sequence of programs	Gray box probing	No	No	No
Build process	Customizing the configuration and building	Yes	No	No
Retargetability issues and machine descriptions	Incremental construction of machine descriptions	No	No	Yes
IR data structures and access mechanisms	Adding passes to massage IRs	No	Yes	Yes
Retargetability mechanism		Yes	Yes	Yes

Workshop Coverage

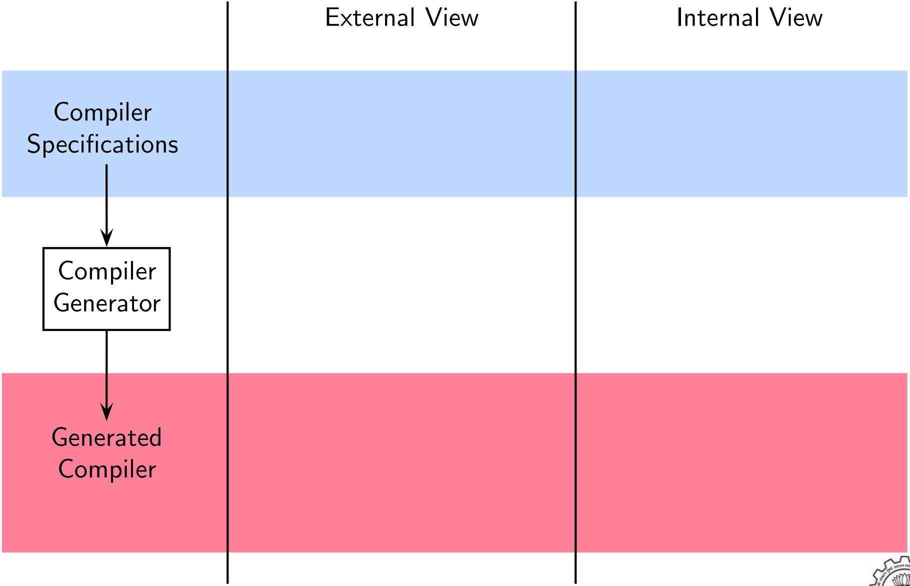


Compiler Specifications

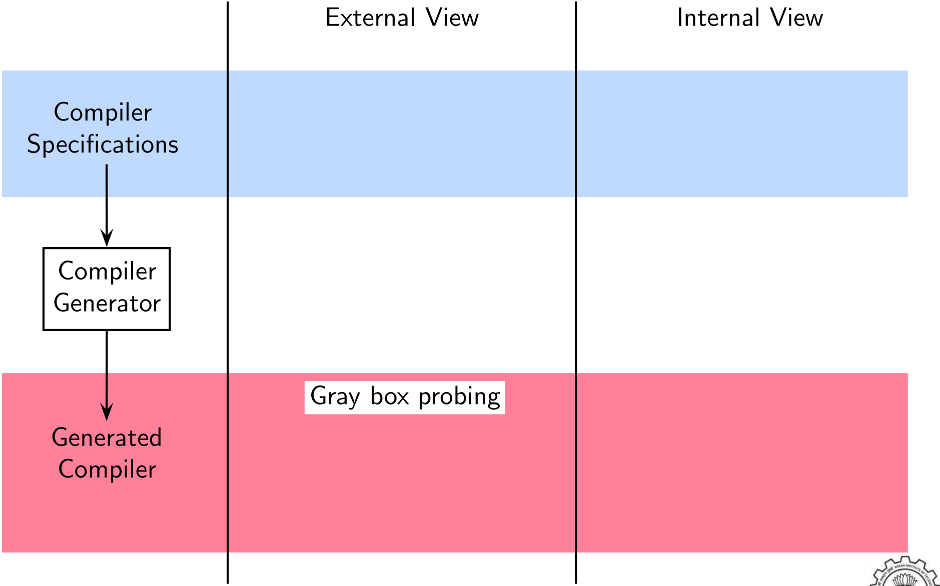
Compiler Generator

Generated Compiler

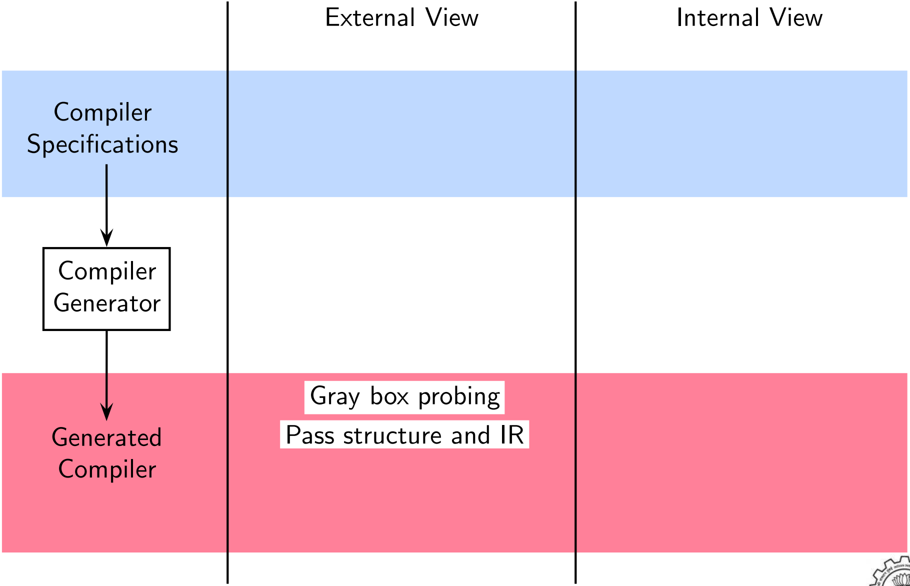
Workshop Coverage



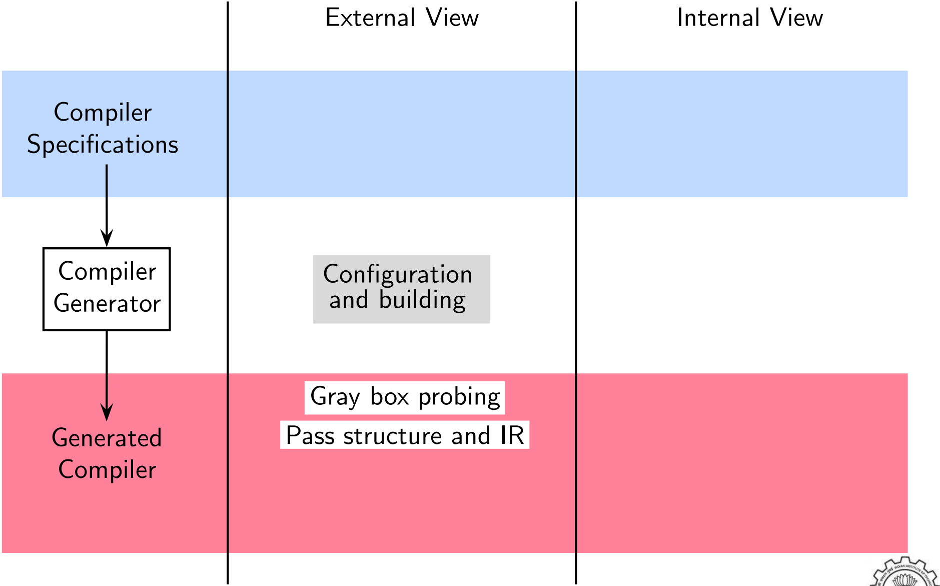
Workshop Coverage



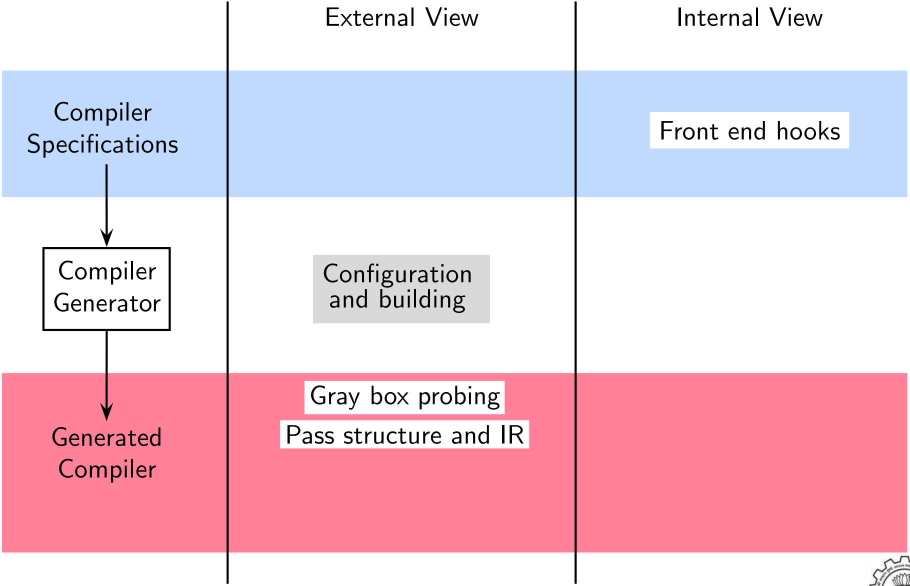
Workshop Coverage



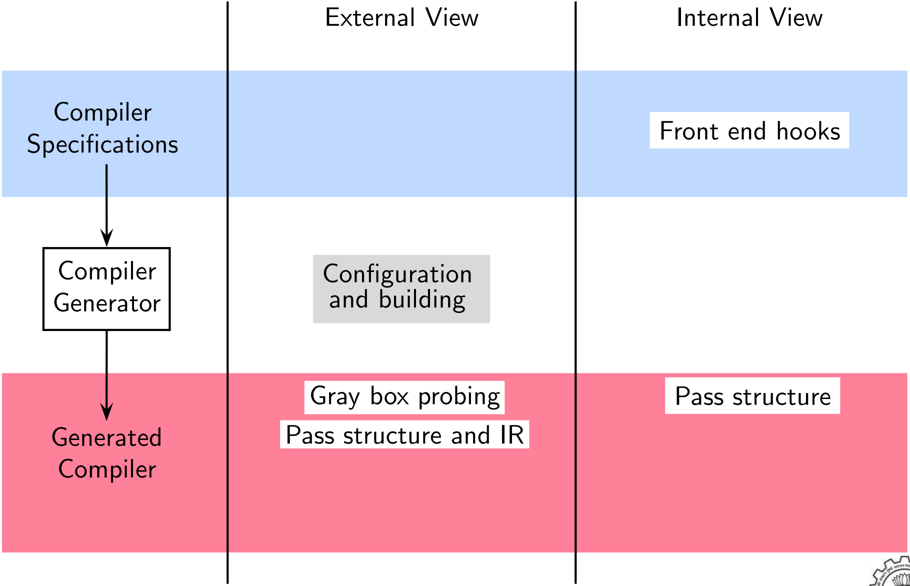
Workshop Coverage



Workshop Coverage

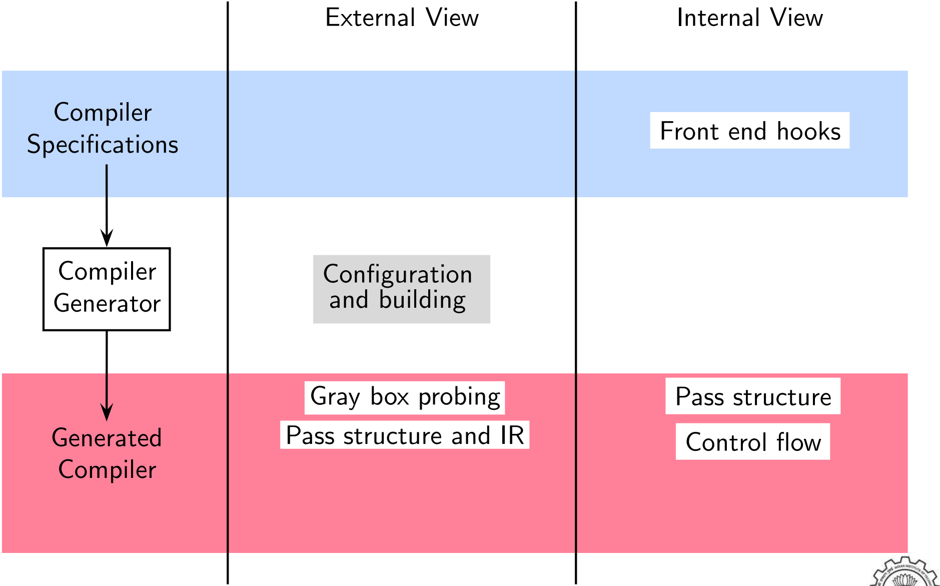


Workshop Coverage



	External View	Internal View
Compiler Specifications		Front end hooks
Compiler Generator	Configuration and building	
Generated Compiler	Gray box probing Pass structure and IR	Pass structure

Workshop Coverage



External View

Internal View

Compiler Specifications

Front end hooks

Compiler Generator

Configuration and building

Generated Compiler

Gray box probing

Pass structure and IR

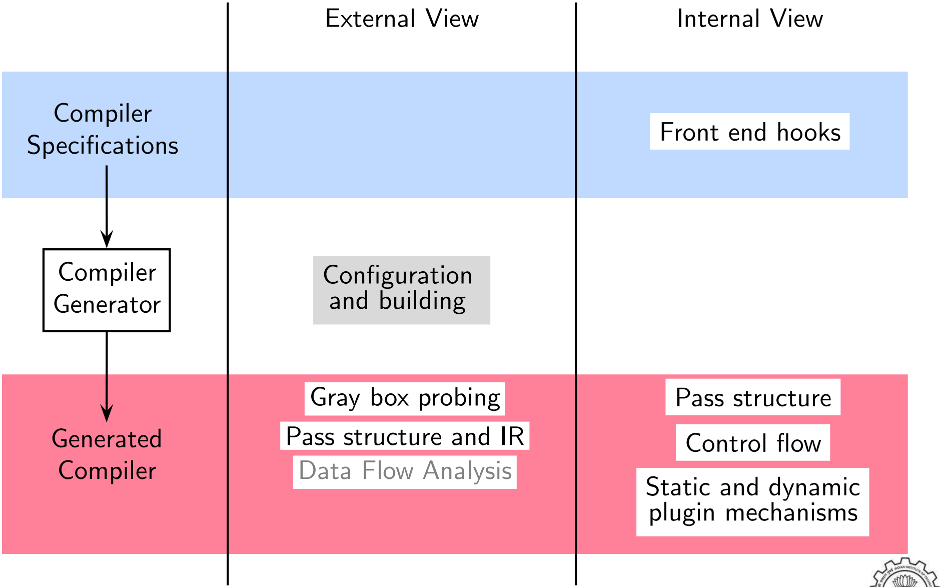
Pass structure

Control flow

Workshop Coverage

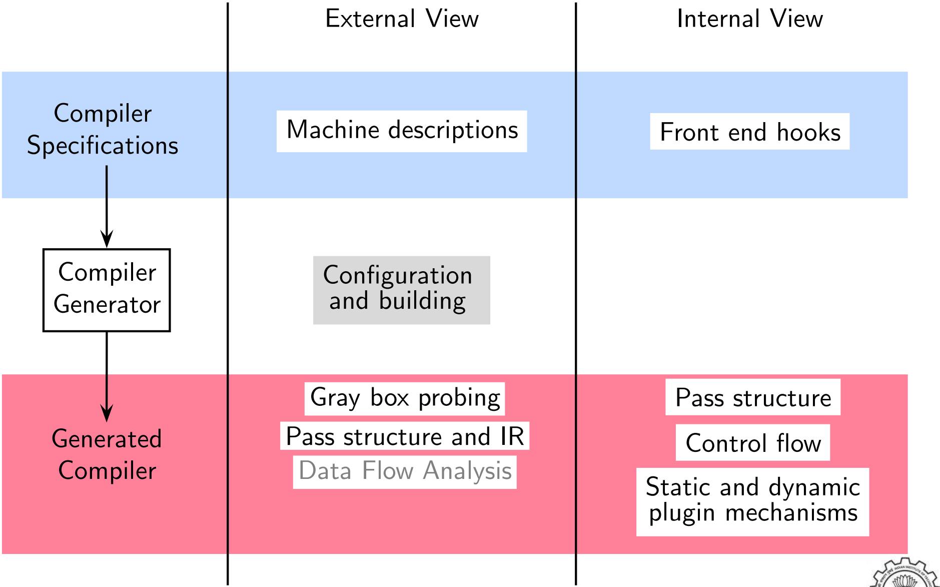
	External View	Internal View
Compiler Specifications		Front end hooks
Compiler Generator	Configuration and building	
Generated Compiler	Gray box probing Pass structure and IR	Pass structure Control flow Static and dynamic plugin mechanisms

Workshop Coverage



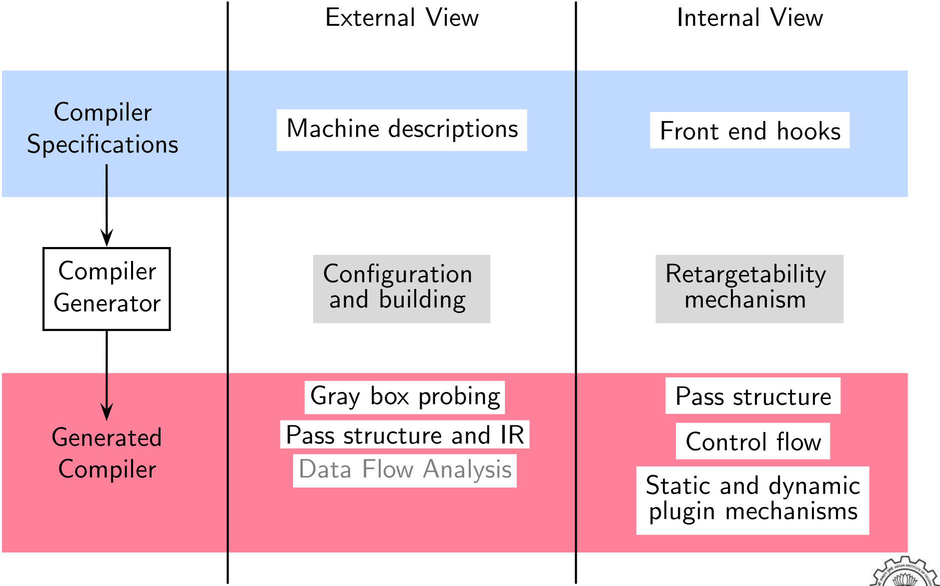
	External View	Internal View
Compiler Specifications		Front end hooks
Compiler Generator	Configuration and building	
Generated Compiler	Gray box probing Pass structure and IR Data Flow Analysis	Pass structure Control flow Static and dynamic plugin mechanisms

Workshop Coverage



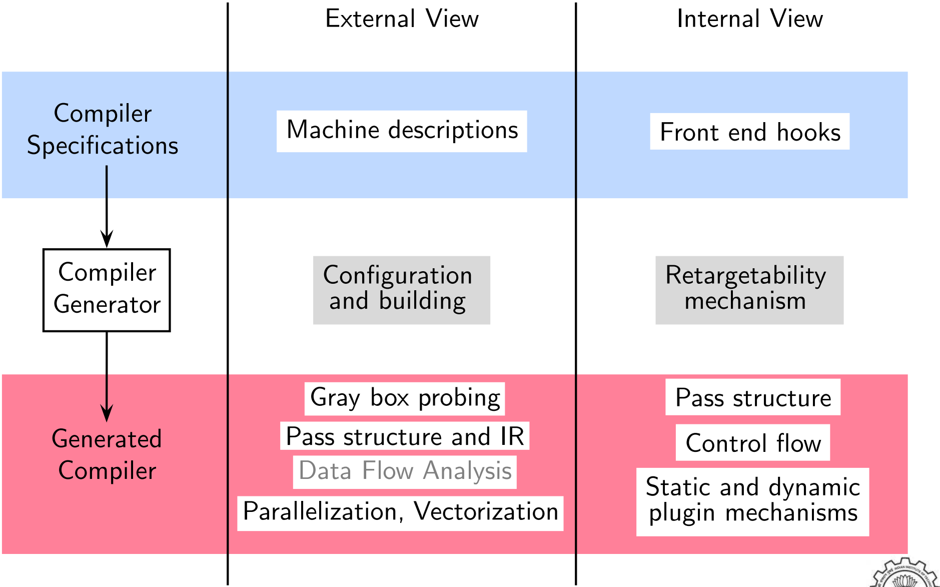
	External View	Internal View
Compiler Specifications	Machine descriptions	Front end hooks
Compiler Generator	Configuration and building	
Generated Compiler	Gray box probing Pass structure and IR Data Flow Analysis	Pass structure Control flow Static and dynamic plugin mechanisms

Workshop Coverage



	External View	Internal View
Compiler Specifications	Machine descriptions	Front end hooks
Compiler Generator	Configuration and building	Retargetability mechanism
Generated Compiler	Gray box probing Pass structure and IR Data Flow Analysis	Pass structure Control flow Static and dynamic plugin mechanisms

Workshop Coverage



	External View	Internal View
Compiler Specifications	Machine descriptions	Front end hooks
Compiler Generator	Configuration and building	Retargetability mechanism
Generated Compiler	Gray box probing Pass structure and IR Data Flow Analysis Parallelization, Vectorization	Pass structure Control flow Static and dynamic plugin mechanisms