CS 715: The Design and Implementation of Gnu Compiler Generation Framework

Uday Khedker

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

January 2011

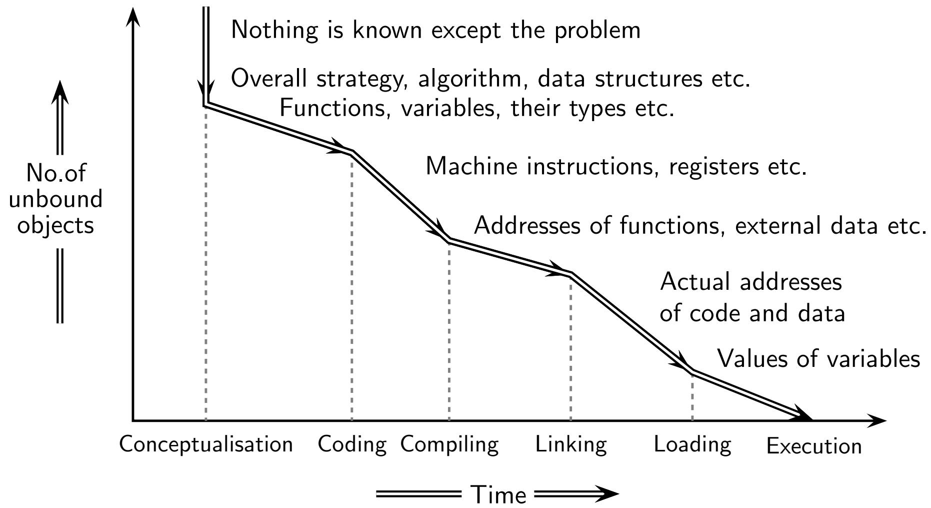
Outline

- An Overview of Compilation
  Introduction, compilation sequence, compilation models
- GCC: The Great Compiler Challenge
  Difficulties in understanding GCC
- Meeting the GCC Challenge: CS 715
  The course plan

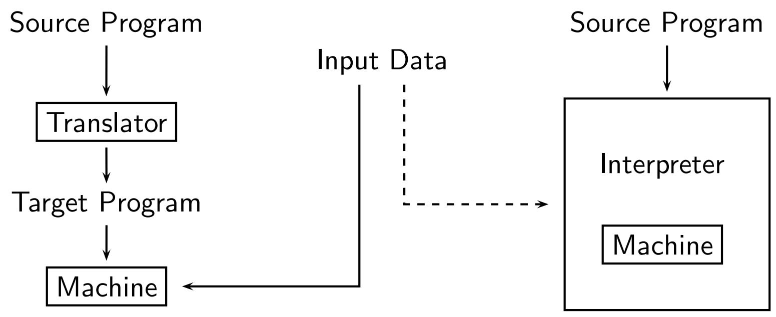
Part 1

Introduction to Compilation

Binding

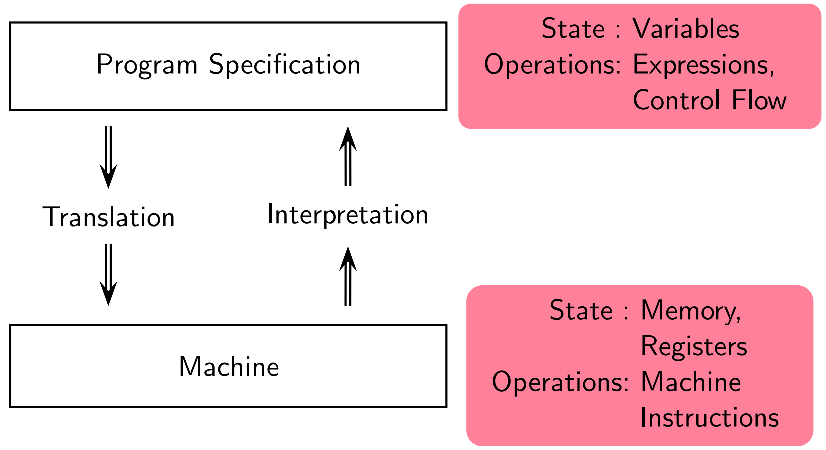


Implementation Mechanisms



Implementation Mechanisms as "Bridges"

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



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
```


High and Low Level Abstractions

Condition

False Part

True Part

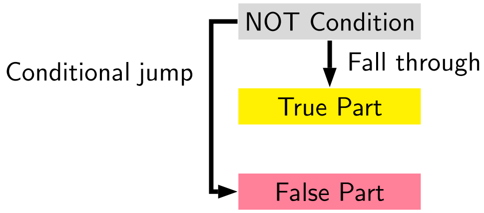
Input C statement

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

High and Low Level Abstractions



Input C statement

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

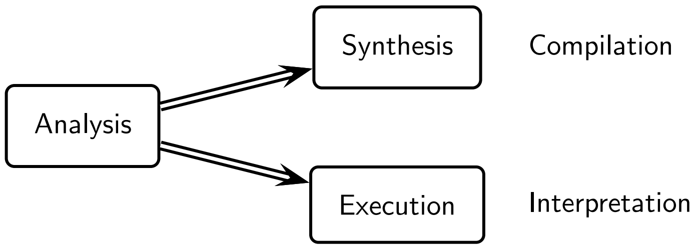
Spim Assembly Equivalent

```
    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
```

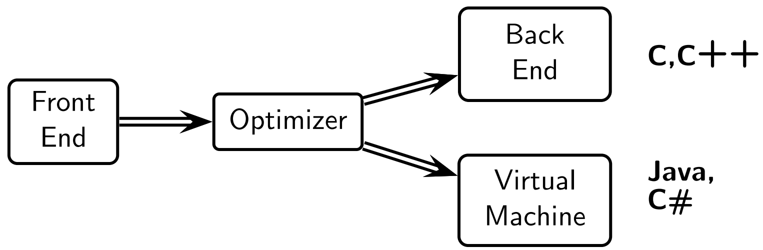

Implementation Mechanisms

- Translation = Analysis + Synthesis
  Interpretation = Analysis + Execution
- Translation Instructions $\Longrightarrow$ Equivalent Instructions
  Interpretation Instructions $\Longrightarrow$ Actions Implied by Instructions

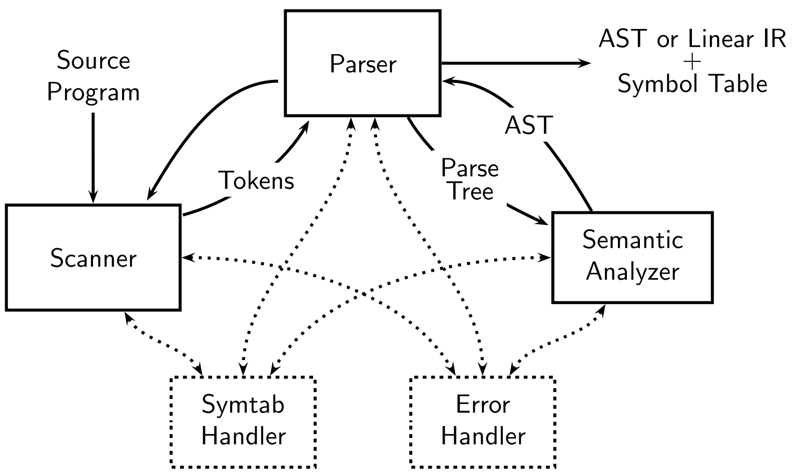
Language Implementation Models



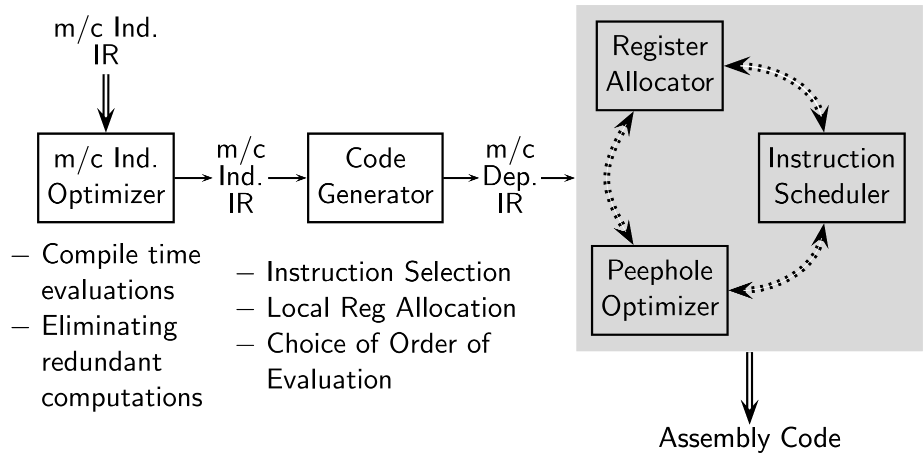
Language Processor Models



Typical Front Ends



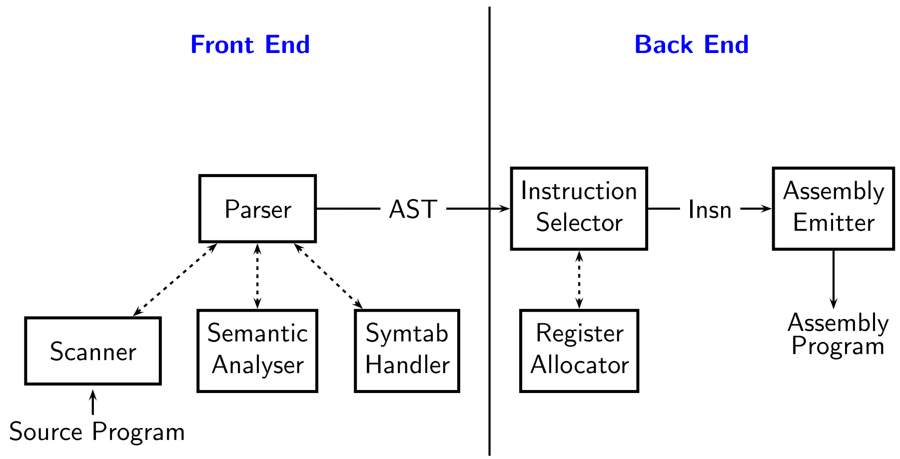
Typical Back Ends



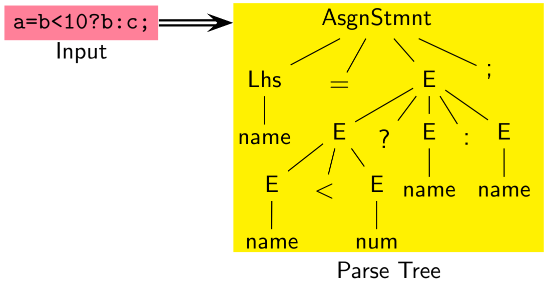
Part 2

An Overview of Compilation Phases

The Structure of a Simple Compiler



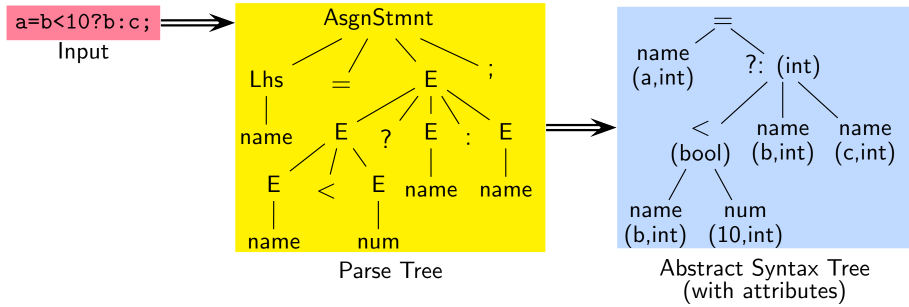
Translation Sequence in Our Compiler: Parsing



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.

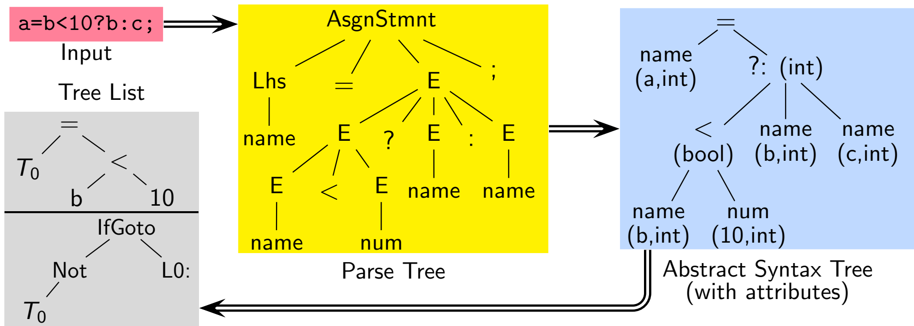
Translation Sequence in Our Compiler: Semantic Analysis

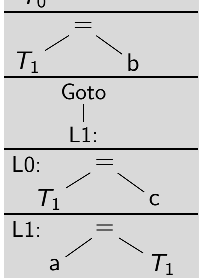


Issues:

- 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

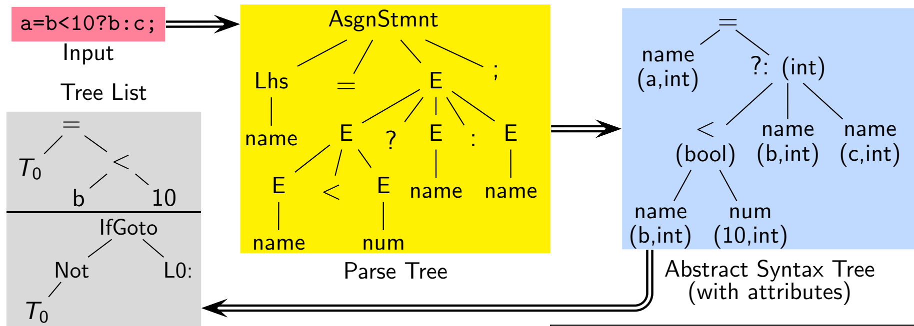


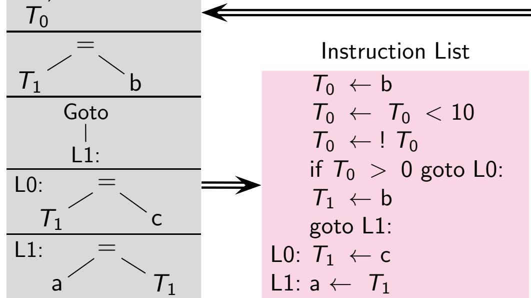


Issues:

- 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

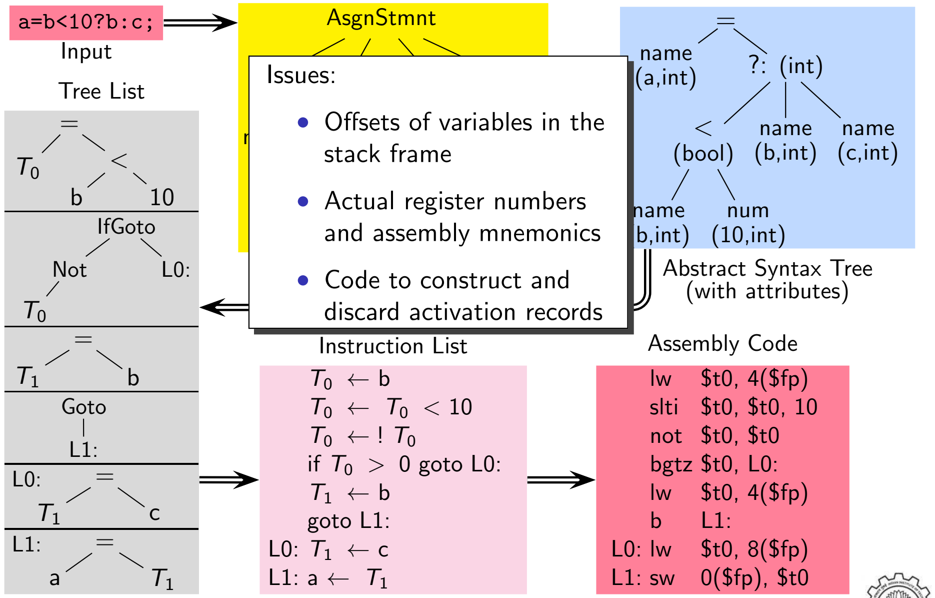




Issues:

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

Translation Sequence in Our Compiler: Emitting Instructions



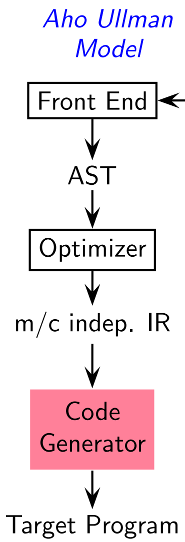
Issues:

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

Part 3

Compilation Models, Instruction Selection, and Retargetability

Compilation Models

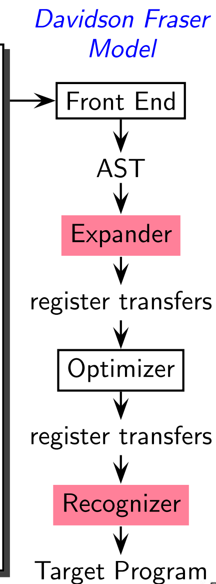


Aho Ullman: Instruction selection

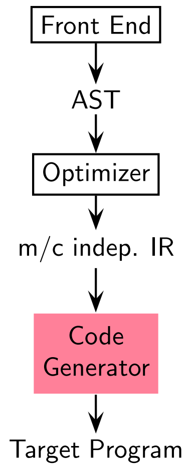
- over optimized IR using
- intelligent tree tiling based algorithms

Davidson Fraser: Instruction selection

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



Retargetability in Aho Ullman Model



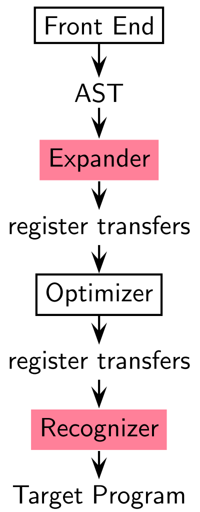
Instruction selection

- over optimized IR using
- intelligent tree tiling based algorithms

Key idea in retargetability:

- Machine independent IR is expressed in the form of trees
- Machine instructions are described in the form of trees
- Trees in the IR are tiled using the instruction trees

Retargetability in Davidson Fraser Model



Instruction selection

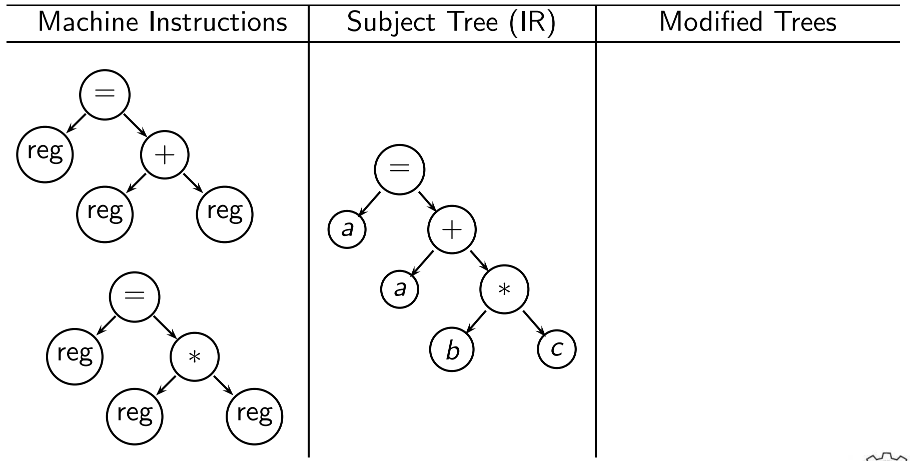
- over AST using
- simple full tree matching based algorithms that generate
- naive code which is
  - machine dependent, and is
  - optimized subsequently

Key idea in retargetability:

- Register transfers are machine specific but
- their form is machine independent

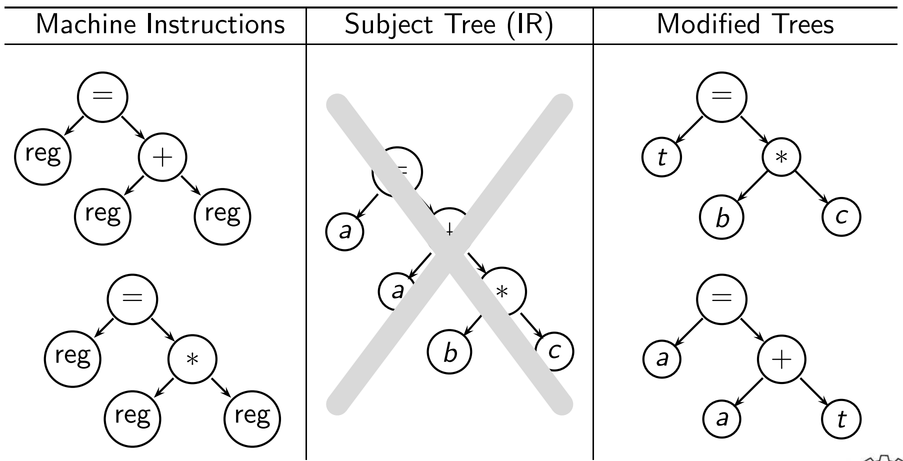
Full Tree Matching (Davidson Fraser Model)

Instructions are viewed as independent non-composable rules



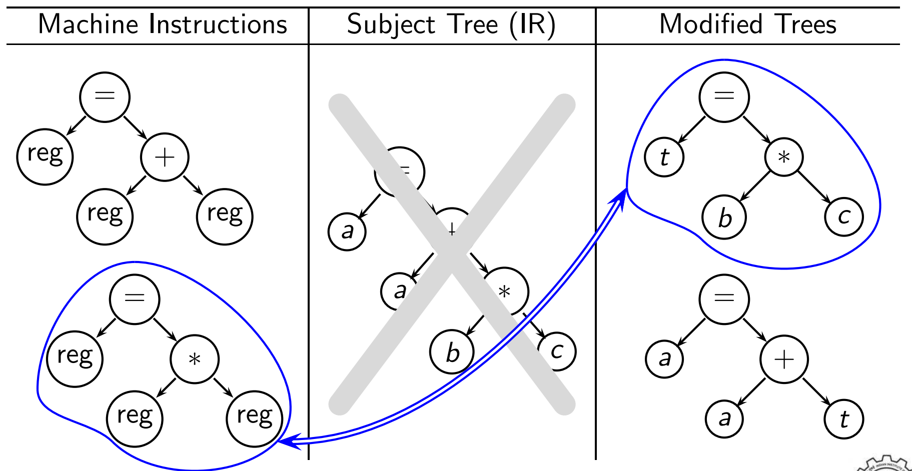
Full Tree Matching (Davidson Fraser Model)

Instructions are viewed as independent non-composable rules



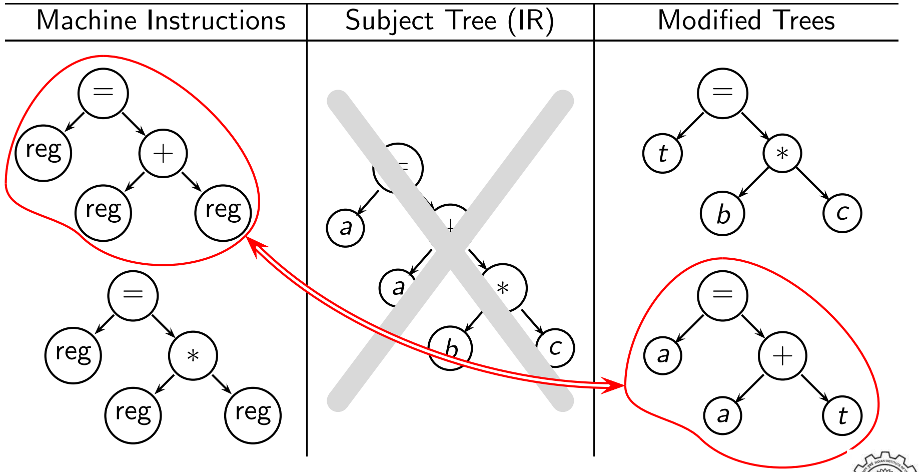
Full Tree Matching (Davidson Fraser Model)

Instructions are viewed as independent non-composable rules



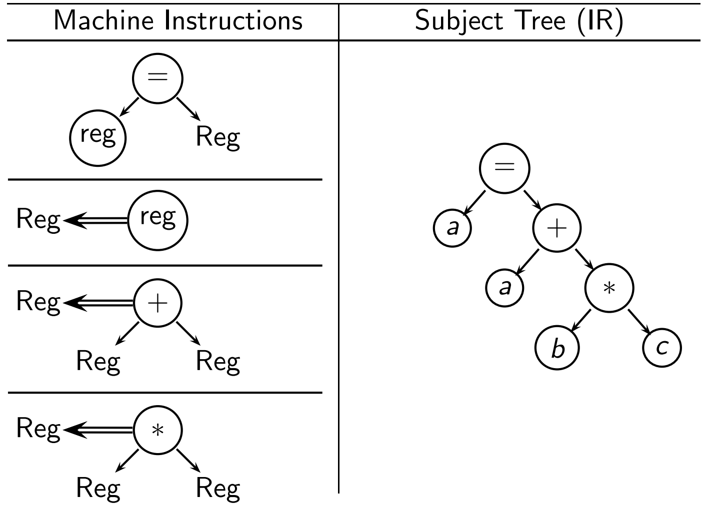
Full Tree Matching (Davidson Fraser Model)

Instructions are viewed as independent non-composable rules



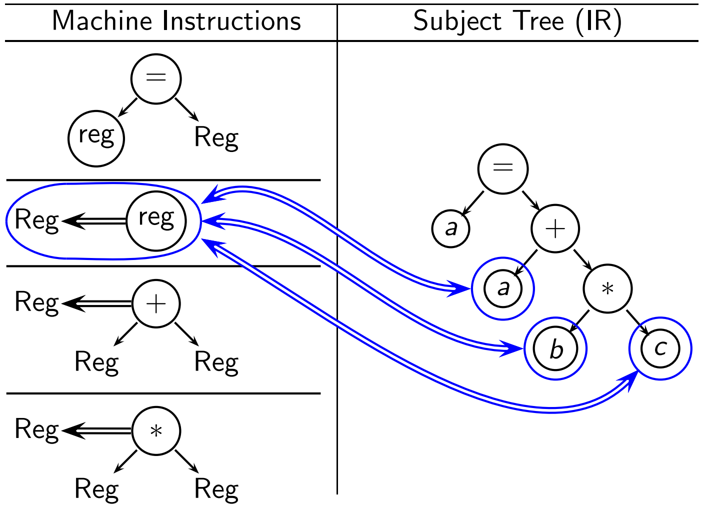
Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules



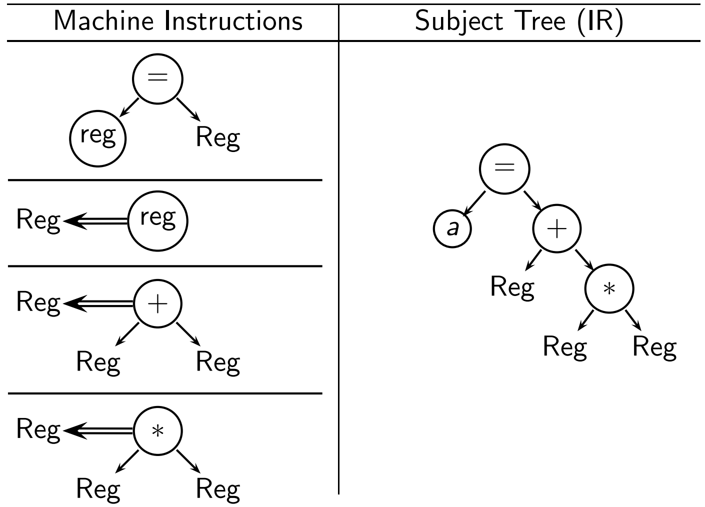
Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules



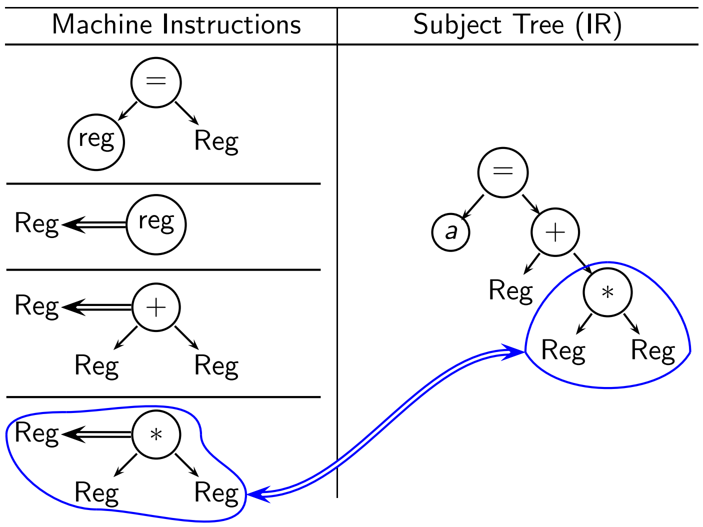
Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules



Tree Tiling (Aho Ullman Model)

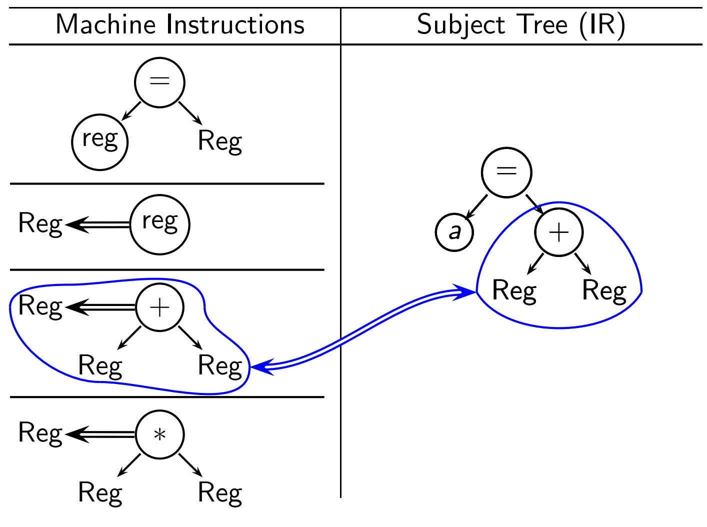
Instructions are viewed as composable rules



CS715 GCC Compiler Collection: Compilation Models, Instruction Selection, and Retargetability 21/55

Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules

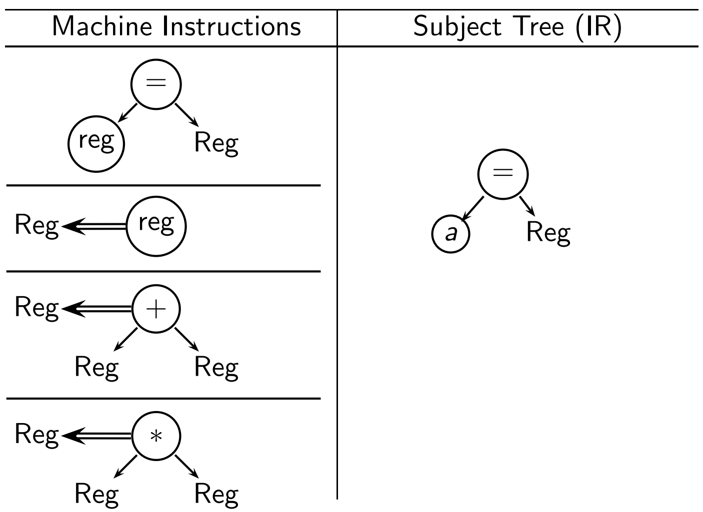


Uday Khedker GRC, IIT Bombay

CS715 GCC Compiler Collection: Compilation Models, Instruction Selection, and Retargetability 21/55

Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules

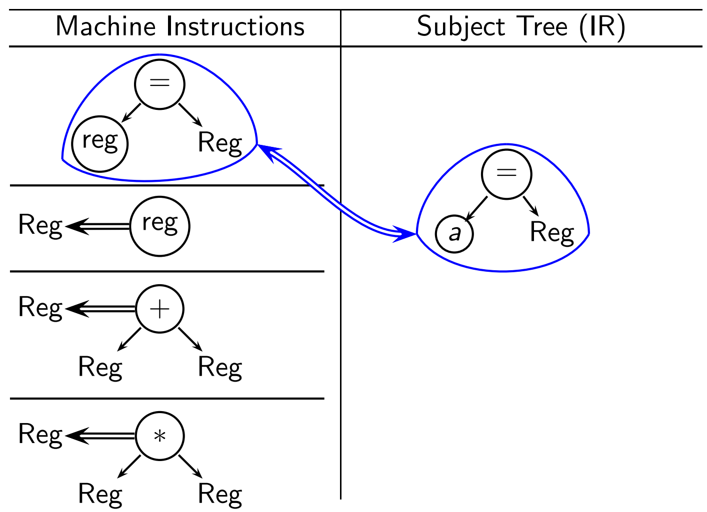


Uday Khedker GRC, IIT Bombay

CS715 GCC Compiler Collection: Compilation Models, Instruction Selection, and Retargetability 21/55

Tree Tiling (Aho Ullman Model)

Instructions are viewed as composable rules

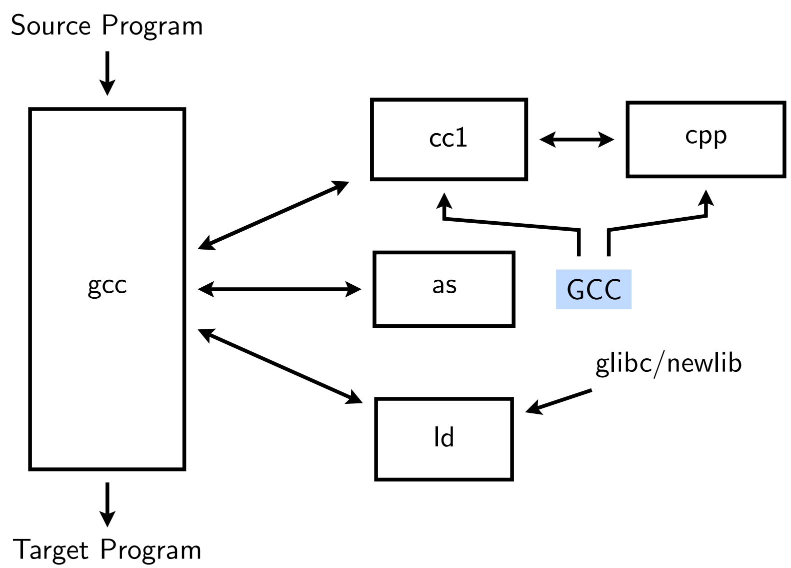


Uday Khedker GRC, IIT Bombay

Part 4

GCC ≡ The Great Compiler Challenge

The Gnu Tool Chain



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

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
    - Input-output of the generated compiler
  - 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

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-4.5.0	4056	65639
gcc-4.6-20101225	4369	70374

- Core size (src/gcc)

	Subdirectories	Files
gcc-4.4.2	257	30163
gcc-4.5.0	283	32723
gcc-4.6-20101225	335	35986

- Machine Descriptions (src/gcc/config)

	Subdirectories	.c files	.h files	.md files
gcc-4.4.2	36	241	426	206
gcc-4.5.0	42	275	478	206
gcc-4.6-20101225	42	269	486	251

`ohcount`: Line Count of gcc-4.4.2

Total: 66139 file(s)

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-4.5.0

Total: 69739 file(s)

Language	Files	Code	Comment	Comment %	Blank	Total
c	16985	1967826	413941	17.4%	391883	2773650
cpp	20813	912618	210084	18.7%	199605	1322307
java	6342	681810	643127	48.5%	169483	1494420
ada	4412	647372	302226	31.8%	222481	1172079
autoconf	79	358996	422	0.1%	55631	415049
html	487	144535	5667	3.8%	31773	181975
make	93	114490	3438	2.9%	14434	132362
fortranfixed	2535	85905	1817	2.1%	11394	99116
assembler	197	45098	10082	18.3%	7528	62708
shell	136	39789	8984	18.4%	5511	54284
scheme	7	13725	1192	8.0%	1524	16441
fortranfree	760	12955	2889	18.2%	1546	17390
objective_c	396	10782	1835	14.5%	2959	15576
automake	64	6388	914	12.5%	994	8296
perl	25	4144	1139	21.6%	739	6022
xslt	20	2805	436	13.5%	563	3804
ocaml	5	2515	540	17.7%	328	3383
python	10	1686	322	16.0%	383	2391
awk	10	1352	372	21.6%	218	1942
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
haskell	49	153	0	0.0%	17	170
emacslisp	1	59	21	26.2%	4	84
matlab	1	5	0	0.0%	0	5

ohcount: Line Count of gcc-4.6-20101225 snapshot

Total: 74787 file(s)

Language	Files	Code	Comment	Comment %	Blank	Total
c	18311	2089300	441364	17.4%	415623	2946287
cpp	21813	977852	227979	18.9%	213239	1419070
java	6342	681938	645505	48.6%	169819	1497262
ada	4601	680002	315946	31.7%	234447	1230395
autoconf	91	397682	513	0.1%	61417	459612
html	446	141275	5391	3.7%	30812	177478
make	99	121013	3615	2.9%	15539	140167
fortranfixed	2852	96084	1920	2.0%	13196	111200
shell	148	47937	10414	17.8%	6566	64917
assembler	209	47015	10287	18.0%	7877	65179
objective_c	815	26409	4669	15.0%	7584	38662
scheme	7	13731	1192	8.0%	1524	16447
fortranfree	806	13667	3104	18.5%	1675	18446
automake	67	9103	971	9.6%	1355	11429
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	1729	396	18.6%	257	2382
python	10	1725	322	15.7%	383	2430
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
haskell	49	153	0	0.0%	17	170
matlab	1	5	0	0.0%	0	5
Total	56846	5408876	1683047	23.7%	1189286	8281209

ohcount: Line Count of gcc-4.4.2/gcc

Total: 30421 file(s)

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	34698[illegible]

ohcount: Line Count of gcc-4.5.0/gcc

Total: 33007 file(s)

Language	Files	Code	Comment	Comment %	Blank	Total
c	14565	1368937	300284	18.0%	305671	1974892
ada	4402	645691	301666	31.8%	221882	1169239
cpp	7984	197798	54719	21.7%	57312	309829
fortranfixed	2453	80403	1768	2.2%	11008	93179
assembler	136	31802	7431	18.9%	4864	44097
autoconf	3	27317	10	0.0%	3876	31203
scheme	7	13725	1192	8.0%	1524	16441
fortranfree	722	12001	2683	18.3%	1446	16130
objective_c	392	10375	1721	14.2%	2838	14934
make	3	5886	1039	15.0%	854	7779
ocaml	5	2515	540	17.7%	328	3383
shell	14	2101	642	23.4%	347	3090
awk	8	1247	299	19.3%	212	1758
perl	2	805	206	20.4%	144	1155
haskell	49	153	0	0.0%	17	170
matlab	1	5	0	0.0%	0	5
Total	30747	2406202	677035	22.0%	613025	369626[illegible]

ohcount: Line Count of gcc-4.6-20101225/gcc snapshot

Total: 36322 file(s)

Language	Files	Code	Comment	Comment %	Blank	Total
c	15638	1452351	319224	18.0%	321806	2093381
ada	4591	678321	315386	31.7%	233848	1227555
cpp	8527	248085	60722	19.7%	66383	375190
fortranfixed	2767	90244	1871	2.0%	12800	104915
assembler	138	31871	7506	19.1%	4882	44259
autoconf	3	28604	12	0.0%	4011	32627
objective_c	810	25860	4492	14.8%	7436	37788
scheme	7	13731	1192	8.0%	1524	16447
fortranfree	768	12713	2893	18.5%	1575	17181
make	4	6124	1070	14.9%	893	8087
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	1624	323	16.6%	251	2198
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	33338	2600787	718924	21.7%	657002	397671[illegible]

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

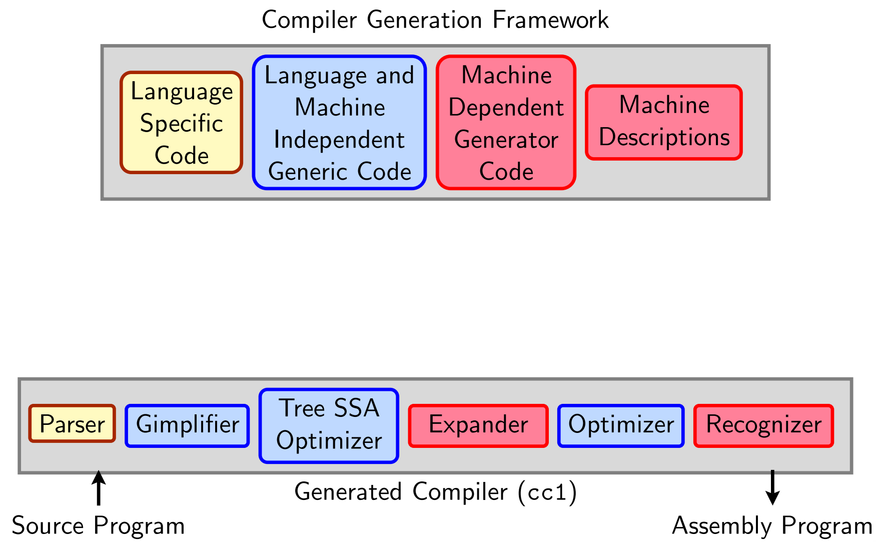
Why is Understanding GCC Difficult?

Deeper reason: GCC is not a *compiler* but a *compiler generation framework*

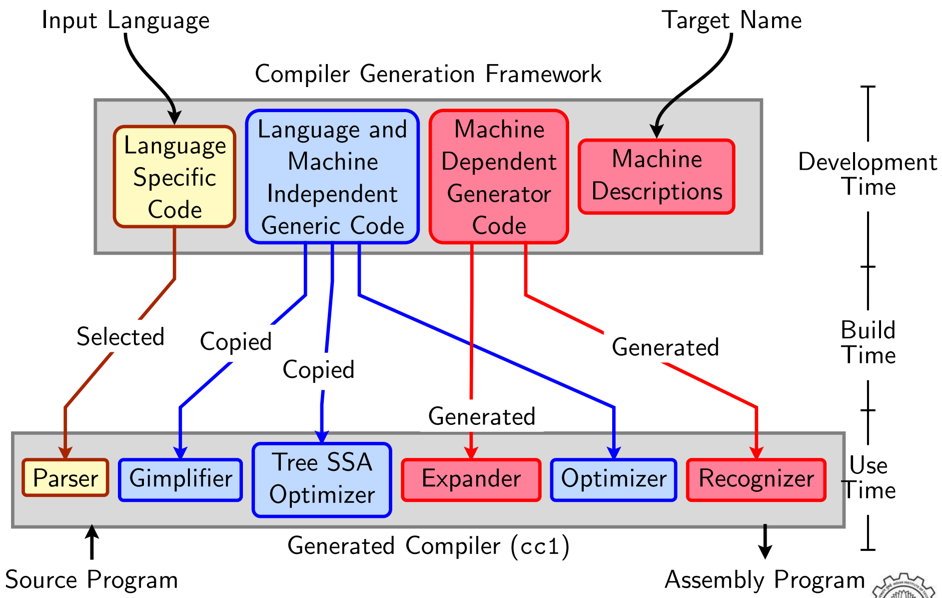
There are two distinct gaps that need to be bridged:

- 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

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;
}
```

- 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 the directory `gcc-4.5.0/gcc` using cscope

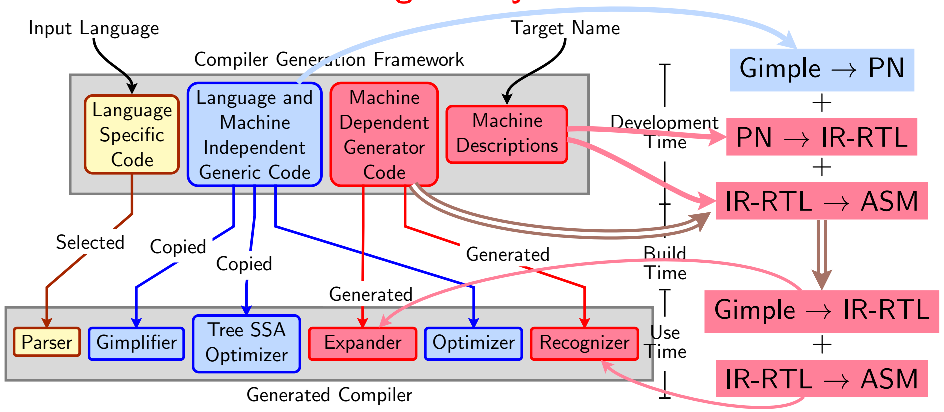
```
  File            Line
0 collect2.c      1111 main (int argc, char **argv)
1 fp-test.c         85 main (void )
2 gcc.c           6803 main (int argc, char **argv)
3 gcov-dump.c       76 main (int argc ATTRIBUTE_UNUSED, char **argv)
4 gcov-iov.c        29 main (int argc, char **argv)
5 gcov.c           355 main (int argc, char **argv)
6 genattr.c         89 main (int argc, char **argv)
7 genattrtab.c    4439 main (int argc, char **argv)
8 genautomata.c   9475 main (int argc, char **argv)
9 genchecksum.c     67 main (int argc, char ** argv)
a gencodes.c        51 main (int argc, char **argv)
b genconditions.c 209 main (int argc, char **argv)
c genconfig.c      261 main (int argc, char **argv)
d genconstants.c    50 main (int argc, char **argv)
e genemit.c        825 main (int argc, char **argv)
f genextract.c     401 main (int argc, char **argv)
```


Another Example of The Generation Related Gap

Locating the `main` function in the directory `gcc-4.5.0/gcc` using cscope

```
  File            Line
g genflags.c       250 main (int argc, char **argv)
h gengenrtl.c      350 main (int argc, char **argv)
i gengtype.c      3694 main (int argc, char **argv)
j genmddeps.c       45 main (int argc, char **argv)
k genmodes.c      1376 main (int argc, char **argv)
l genopinit.c      469 main (int argc, char **argv)
m genoutput.c     1023 main (int argc, char **argv)
n genpeep.c        353 main (int argc, char **argv)
o genpreds.c      1404 main (int argc, char **argv)
p genrecog.c      2722 main (int argc, char **argv)
q lto-wrapper.c    412 main (int argc, char *argv[])
r main.c            33 main (int argc, char **argv)
s mips-tdump.c    1393 main (int argc, char **argv)
t mips-tfile.c     655 main (void )
u mips-tfile.c    4695 main (int argc, char **argv)
v tlink.c           61 const char *main;
```

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

Files	i386	mips
*.md	35766	12930
*.c	28643	12572
*.h	15694	5105

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

Part 5

Meeting the GCC Challenge

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

What is Gray Box Probing of GCC?

- Black Box probing:
  Examining only the input and output relationship of a system
- White Box probing:
  Examining internals of a system for a given set of inputs
- Gray Box probing:
  Examining input and output of various components/modules
  - Overview of translation sequence in GCC
  - Overview of intermediate representations
  - Intermediate representations of programs across important phases

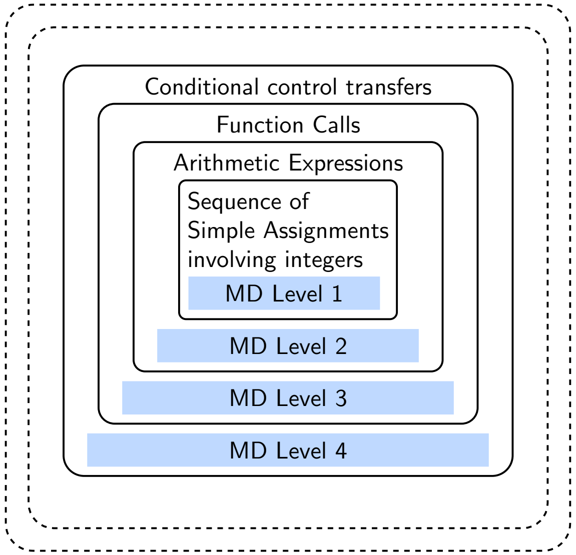
Customizing the Configuration and Build Process

- Creating only cc1
- Creating bare metal cross build
  Complete tool chain without OS support
- Creating cross build with OS support

Incremental Construction of Machine Descriptions

- Define different levels of source language
- Identify the minimal set of features in the target required to support each level
- Identify the minimal information required in the machine description to support each level
  - Successful compilation of any program, and
  - correct execution of the generated assembly program
- Interesting observations
  - It is the increment in the source language which results in understandable increments in machine descriptions rather than the increment in the target architecture
  - If the levels are identified properly, the increments in machine descriptions are monotonic

Incremental Construction of Machine Descriptions



Adding Passes to Massage IRs

- Understanding the pass structure
- Understanding the mechanisms of traversing a call graph and a control flow graph
- Understanding how to access the data structures of IRs
- Simple exercises such as:
  - Count the number of copy statements in a program
  - Count the number of variables declared "const" in the program
  - Count the number of occurances of arithmatic operators in the program
  - Count the number of references to global variables in the program

CS 715 Coverage

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

CS 715 Coverage

- An external view of GCC:
  - Configuration and building
  - Gray box probing of GCC
  - Introduction to Gimple and RTL IRs
  - Parallelization and vectorization in GCC
  - Introduction to GCC Machine Decriptions
- An internal view of GCC: Walking the maze of GCC source code
  - Control flow and plugin structure of the core compiler,
  - Mechanisms for hooking up front ends, IR passes, and back ends
  - Examining and manipulating Gimple and RTL IRs
  - Design and implementation of GDFA
  - Machine descriptions and retargetability mechanism

CS 715 Philosophy

Twin goals:

- *Learning how to learn GCC*

  Our focus will be on
  - giving you some core information
  - showing you how to discover more information
- *Striking a balance between theory and practice*

  Our focus will be on showing you how to
  - discover concepts in a large code base and build abstractions
  - take concepts and update a large code base
  - relate the class room concepts of complilers to an industry strength compiler

CS 715 Pedagogy

- Introductory lecture for each topic followed by lab work
- Many the lecture hours will be used as lab hours
  - We will meet a one place (NSL? GRC?)
    You are expected to do ssh to your own machine
  - Class room as labs using your laptops?
- Tools
  shell, make, cscope, ctags, gdb/ddd, screen, spim (mips simulator)

CS 715 Lab Work

- Lab exercises:
  - Pre-defined experiments
  - Ungraded
- Assignments: Graded lab work
  - Implementation to modify gcc
  - Graded
  - To be submitted in about a couple of weeks or 10 days
- Projects
  - Specific Study + Implementation
  - Graded
  - To be submitted in about a couple of months
  - Possible topics
    Extensions of GDFA, MD rewriting with newer constructs, Detailing the retargetability mechanism, MD parsers, garbage collection, MELT, investigating specific optimizations (such as constant splitting)

CS 715 Assessment Scheme

- No written examination
- Marks for lab work

Head	Number	Weightage
Assignments	4	$4 \times 15 = 60$
Project	1	40
Total		100

- Assignments need not be same for all students
  However, they will be comparable and students would have the choice of opting for a particular assignment