GCC Configuration and Building

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

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

July 2008

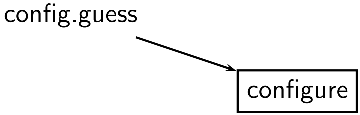
Part 1

Configuration and Building

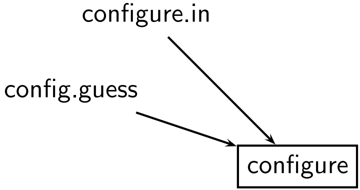
Configuring GCC

configure

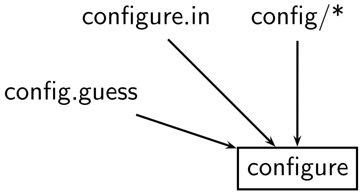
Configuring GCC



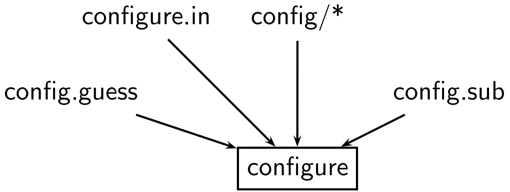
Configuring GCC



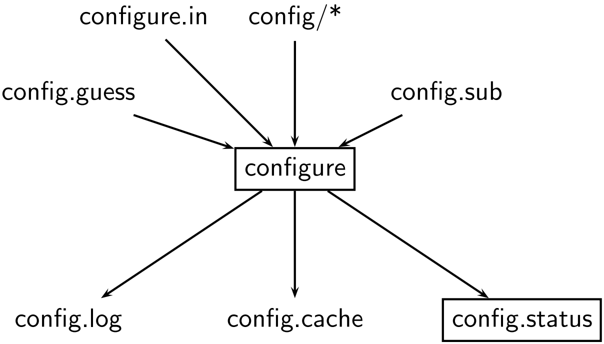
Configuring GCC



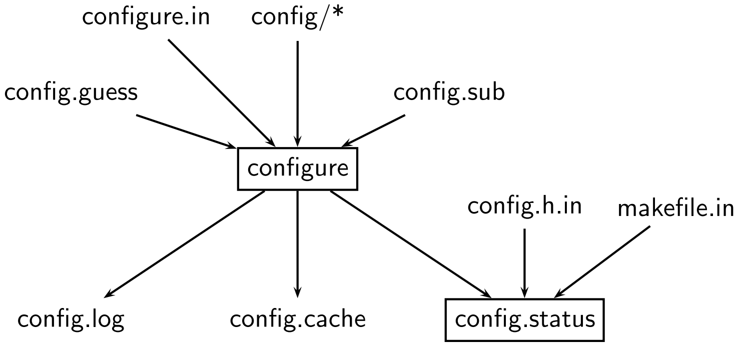
Configuring GCC



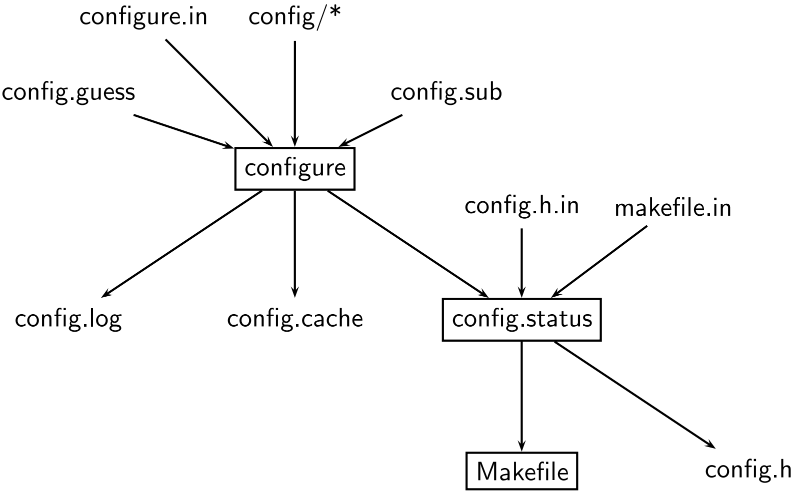
Configuring GCC



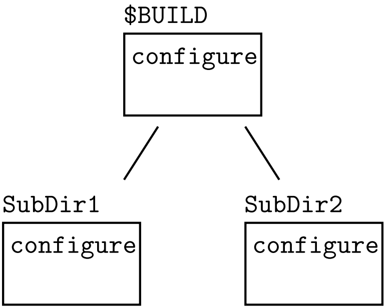
Configuring GCC



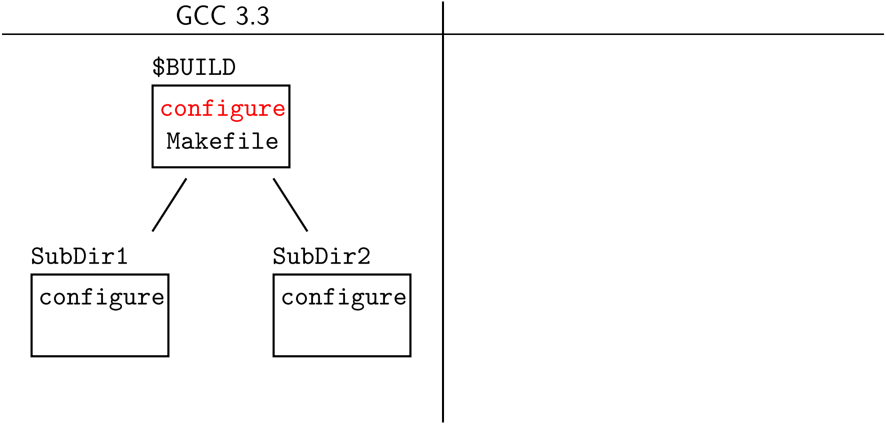
Configuring GCC



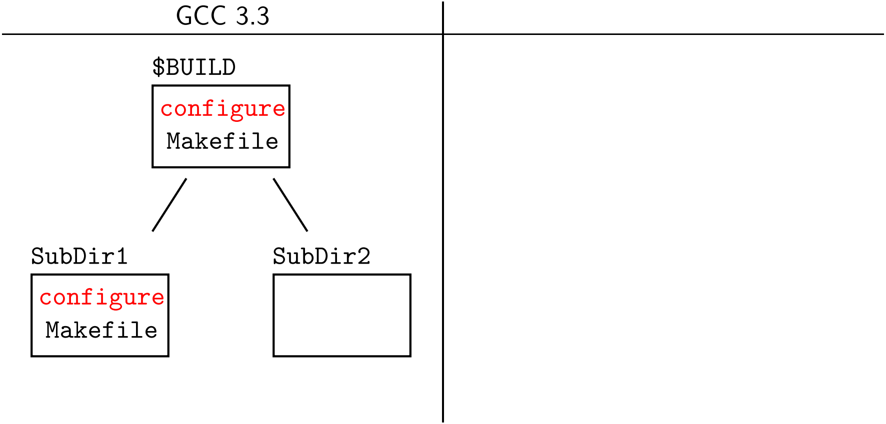
Alternatives in Configuration

GCC 3.3	
	

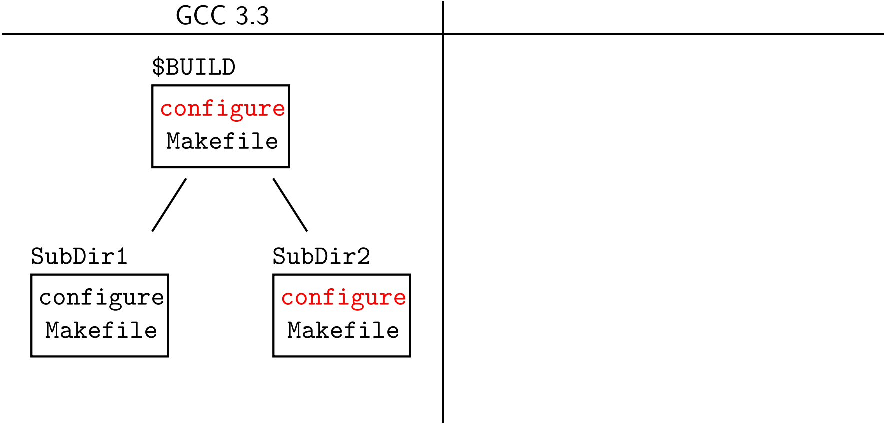
Alternatives in Configuration



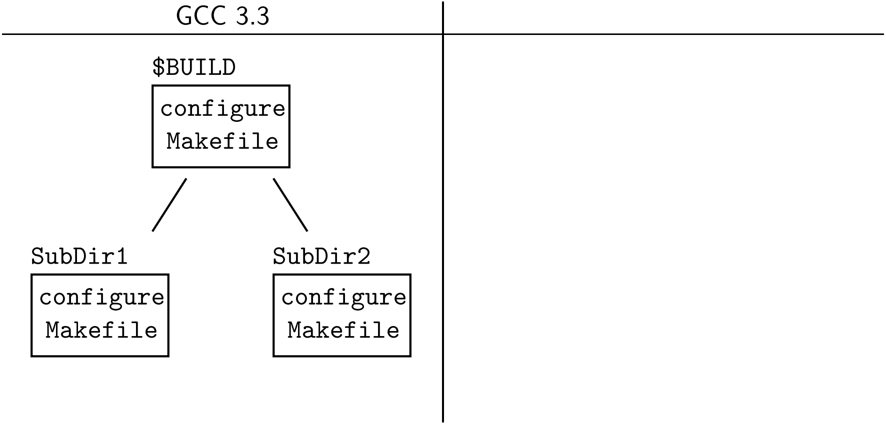
Alternatives in Configuration



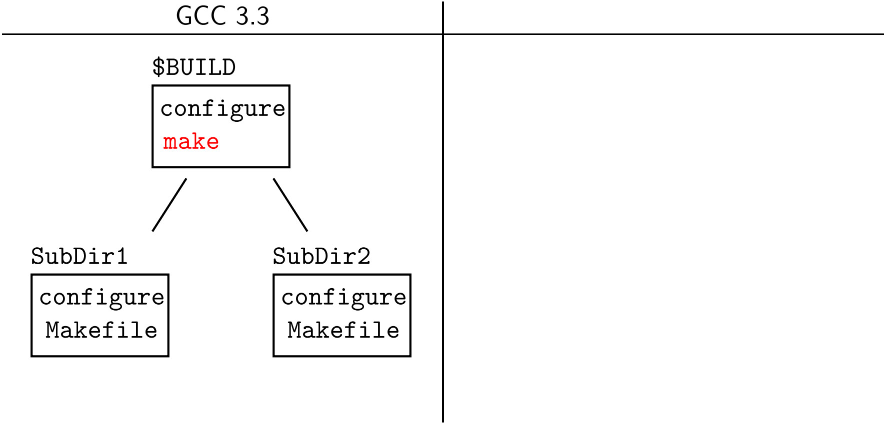
Alternatives in Configuration



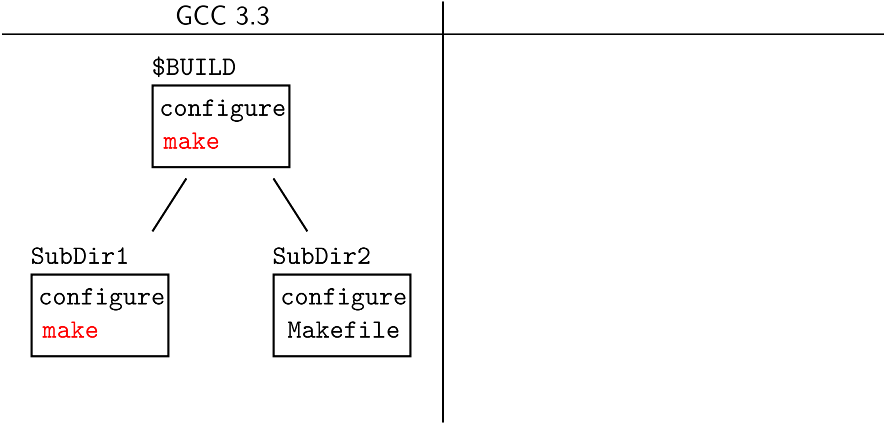
Alternatives in Configuration



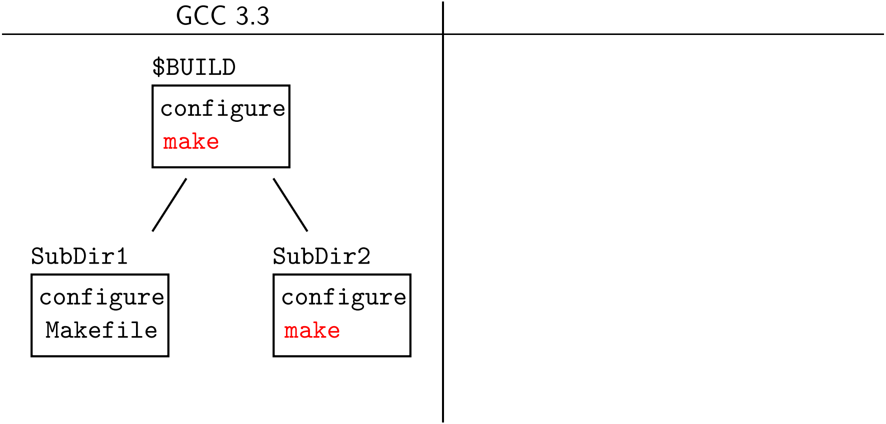
Alternatives in Configuration



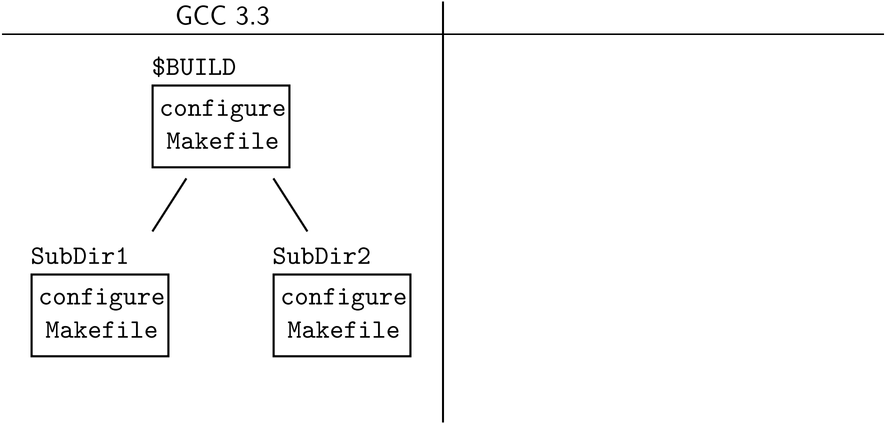
Alternatives in Configuration



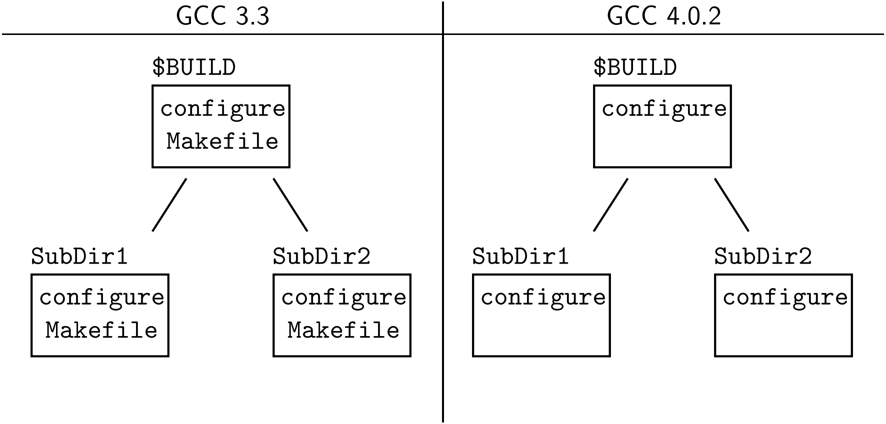
Alternatives in Configuration



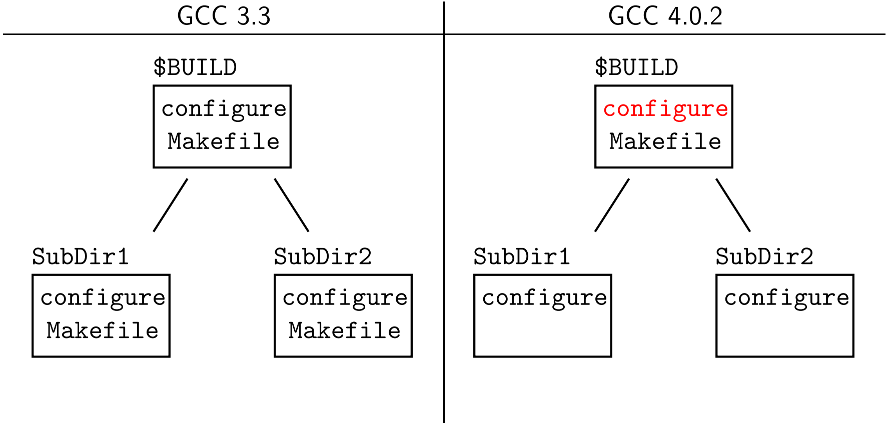
Alternatives in Configuration



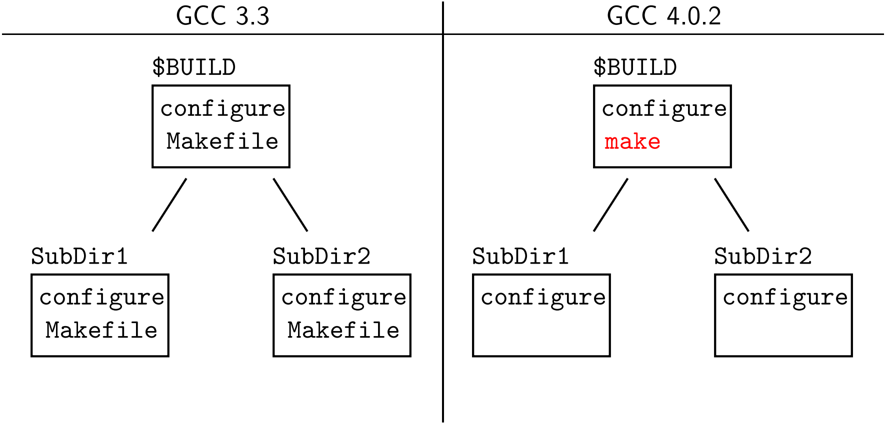
Alternatives in Configuration



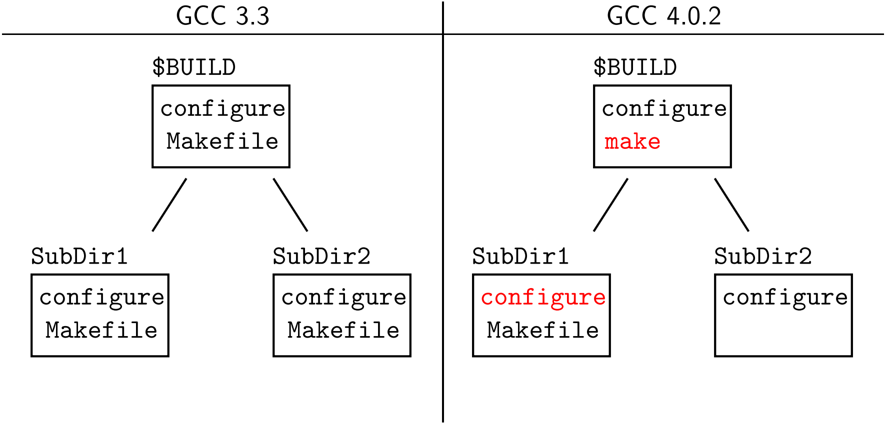
Alternatives in Configuration



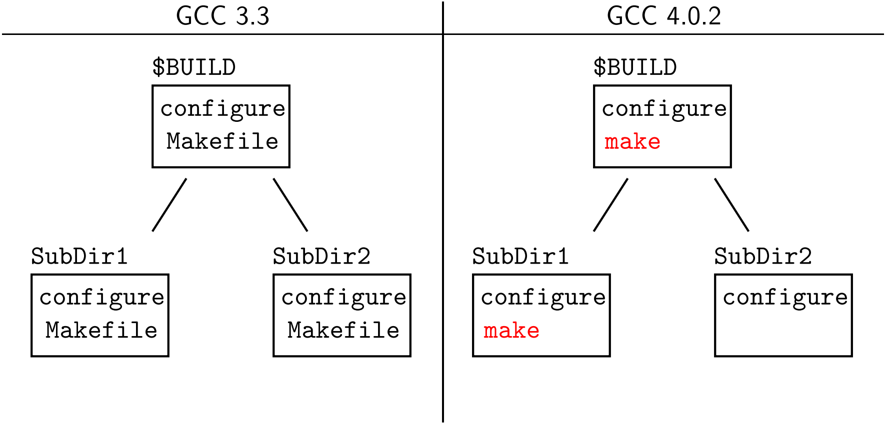
Alternatives in Configuration



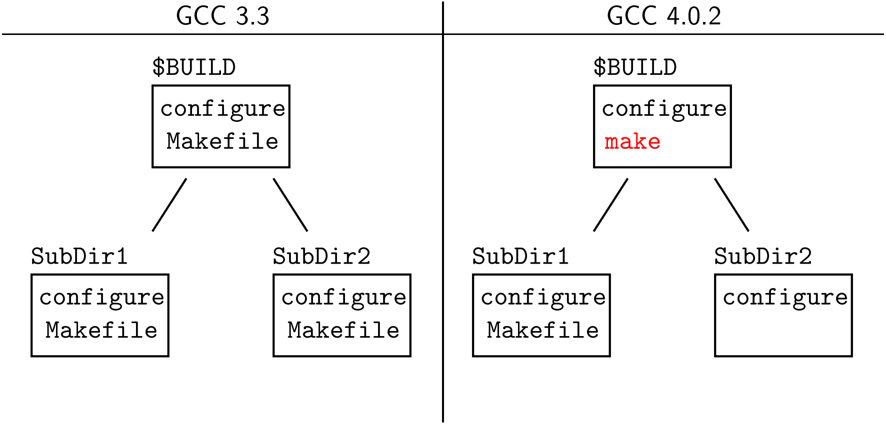
Alternatives in Configuration



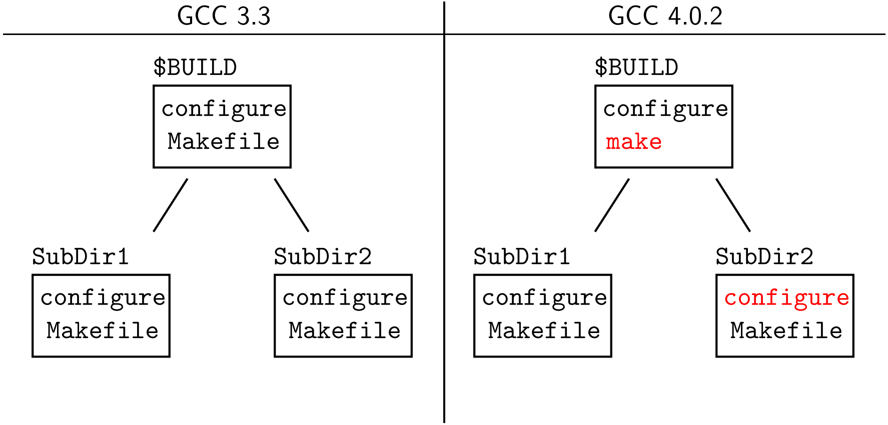
Alternatives in Configuration



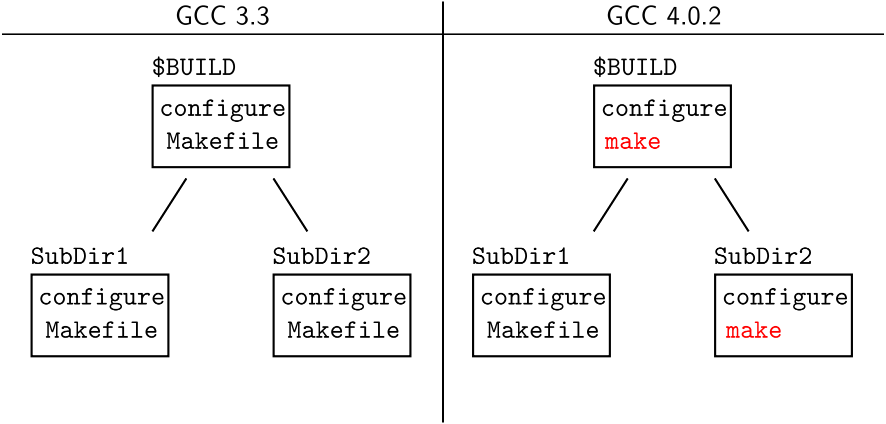
Alternatives in Configuration



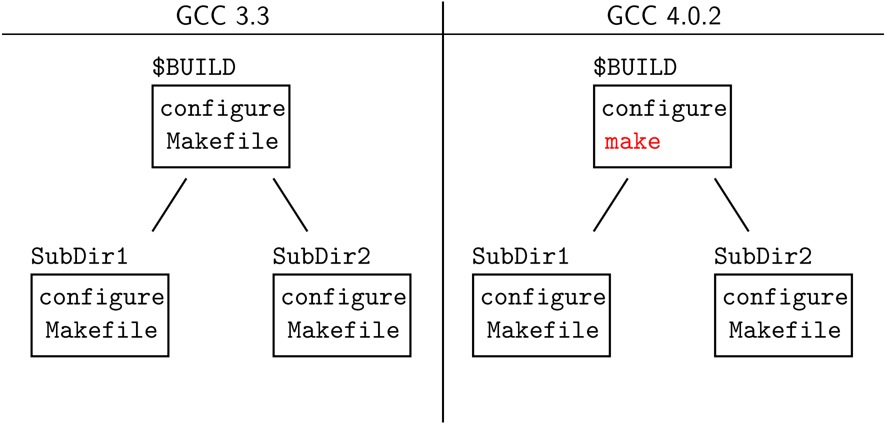
Alternatives in Configuration



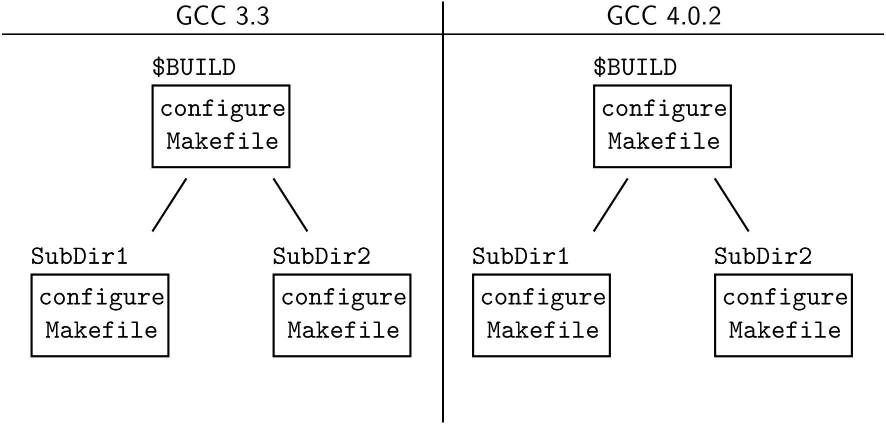
Alternatives in Configuration



Alternatives in Configuration



Alternatives in Configuration



Steps in Configuration and Building

Usual Steps	
• Download and untar the source	

Steps in Configuration and Building

Usual Steps	
• Download and untar the source • `cd $SOURCE`	

Steps in Configuration and Building

Usual Steps	
• Download and untar the source • `cd $SOURCE` • `./configure`	

Steps in Configuration and Building

Usual Steps	
• Download and untar the source • `cd $SOURCE` • `./configure` • `make`	

Steps in Configuration and Building

Usual Steps	
• Download and untar the source • `cd $SOURCE` • `./configure` • `make` • `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source • `cd $SOURCE` • `./configure` • `make` • `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source • `cd $SOURCE` • `./configure` • `make` • `make install`	• Download and untar the source

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	
• `make`	
• `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	• `$SOURCE/configure`
• `make`	
• `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	• `$SOURCE/configure`
• `make`	• `make`
• `make install`	

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	• `$SOURCE/configure`
• `make`	• `make`
• `make install`	• `make install`

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	• `$SOURCE/configure`
• `make`	• `make`
• `make install`	• `make install`
	$SOURCE and $BUILD must be distinct!

Steps in Configuration and Building

Usual Steps	Steps in GCC
• Download and untar the source	• Download and untar the source
• `cd $SOURCE`	• `cd $BUILD`
• `./configure`	• `$SOURCE/configure`
• `make`	• `make`
• `make install`	• `make install`
	$SOURCE and $BUILD must be distinct!

GCC generates a large part of source code during configuration!

Some Interesting Facts about GCC 4.0.2

Pristine compiler sources (downloaded tarball)

Lines of C code	1098306
Lines of MD code	217888
Lines of total code	1316194
Total Authors (approx)	63
Backend directories	34

For the targetted (= pristine + generated) C compiler

Total lines of code	810827
Total lines of pure code	606980
Total pure code WITHOUT #includes	602351
Total number of #include directives	4629
Total #included files	336

Some Interesting i386 MD Facts

General information

Number of .md files	8
Number of C files	72

Realistic code size information (excludes comments)

Total lines of code	47290
Total lines of .md code	23566
Total lines of header code	9986
Total lines of C code	16961

Other Facts about the Size of GCC

	GCC 4.0.2	GCC 4.3.0
Compressed tar file (.tar.bz2)	30.3 MB	6.6 MB
Uncompressed source	378 MB	511 MB

The size of $BUILD directory is usually larger than $SOURCE and depends on the target specification.

Building a Compiler: General issues I

Some Terminology

- The sources of a compiler are compiled (i.e. built) on machine X
  X is called as the Build system
- The built compiler runs on machine Y
  Y is called as the Host system
- The compiler compiles code for target Z
  Z is called as the Target system
- Note: The built compiler itself runs on the Host machine and generates executables that run on Target machine!!!

Building a Compiler: General issues II

Some Definitions

Note: The built compiler itself runs on the Host machine and generates executables that run on Target machine!!!

A few interesting permutations of X, Y and Z are:

$X = Y = Z$	Native build
$X = Y \neq Z$	Cross compiler
$X \neq Y \neq Z$	Canadian Cross compiler

Example

Native i386: built on i386, hosted on i386, produces i386 code.
Sparc cross on i386: built on i386, hosted on i386, produces Sparc code.

Building a Compiler

Bootstrapping

A compiler is just another program

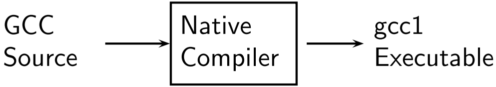
It is improved, bugs are fixed and newer versions are released

To build a new version given a built old version:

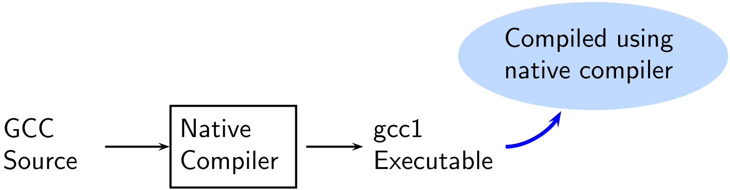
1. Stage 1: Build the new compiler using the old compiler
2. Stage 2: Build another new compiler using compiler from stage 1
3. Stage 3: Build another new compiler using compiler from stage 2
   Stage 2 and stage 3 builds must result in identical compilers

⇒ Building cross compilers stops after Stage 1!

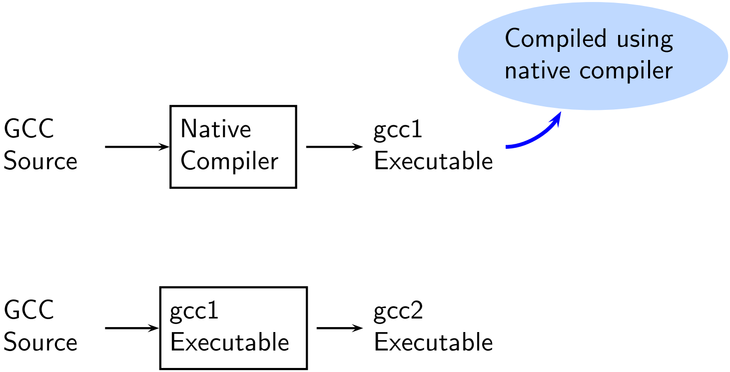
A Native Build



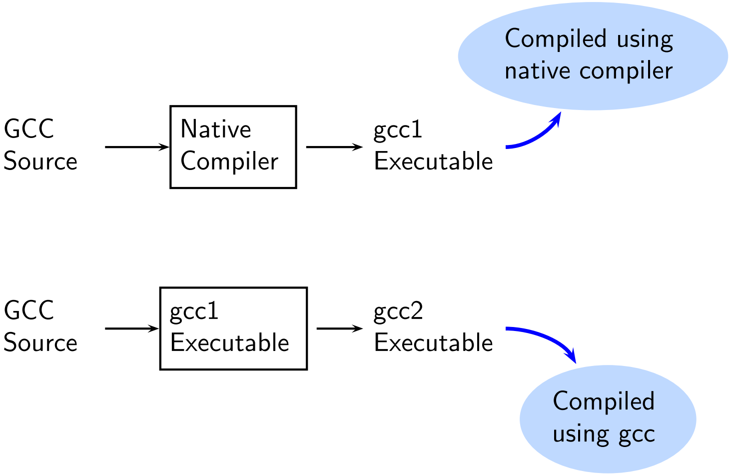
A Native Build



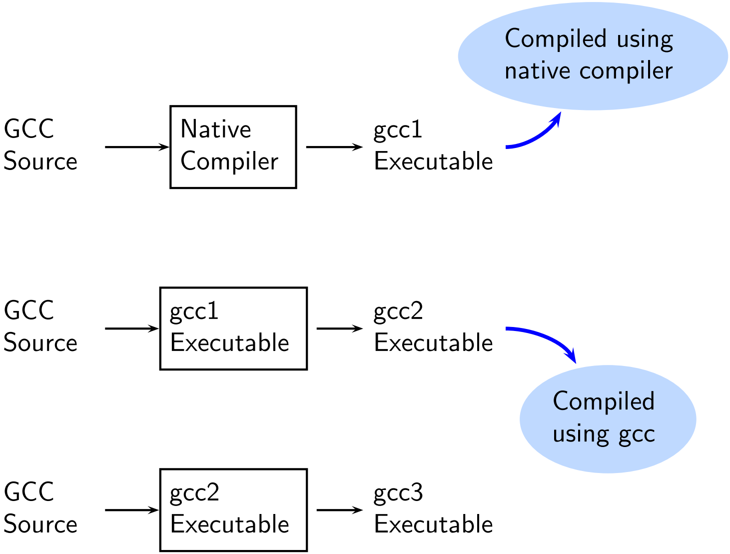
A Native Build



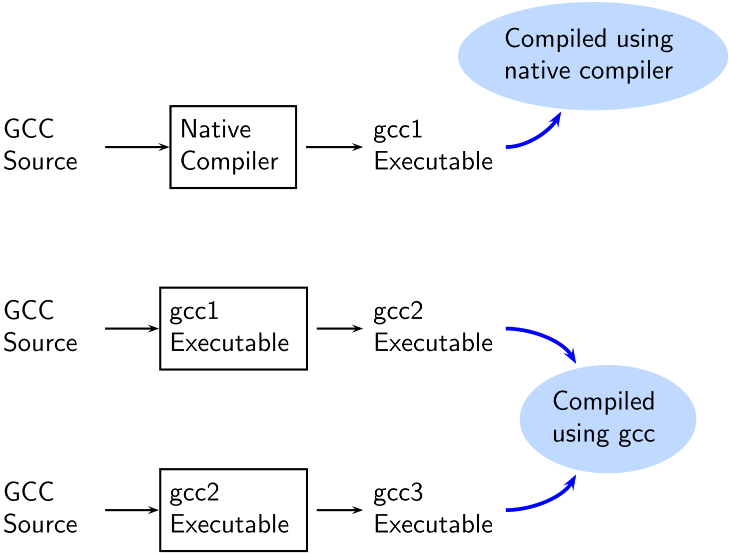
A Native Build



A Native Build

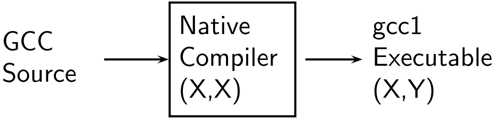


A Native Build



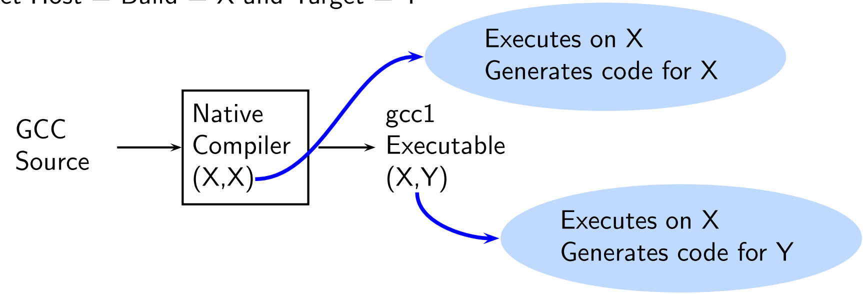
A Cross Build

Let Host = Build = X and Target = Y



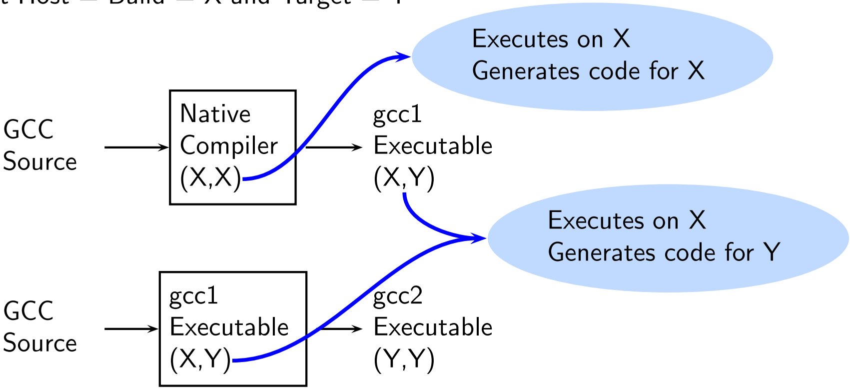
A Cross Build

Let Host = Build = X and Target = Y



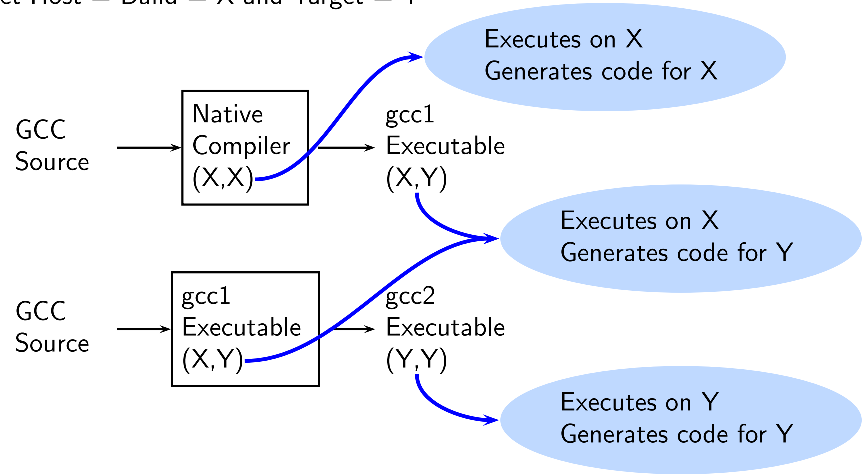
A Cross Build

Let Host = Build = X and Target = Y



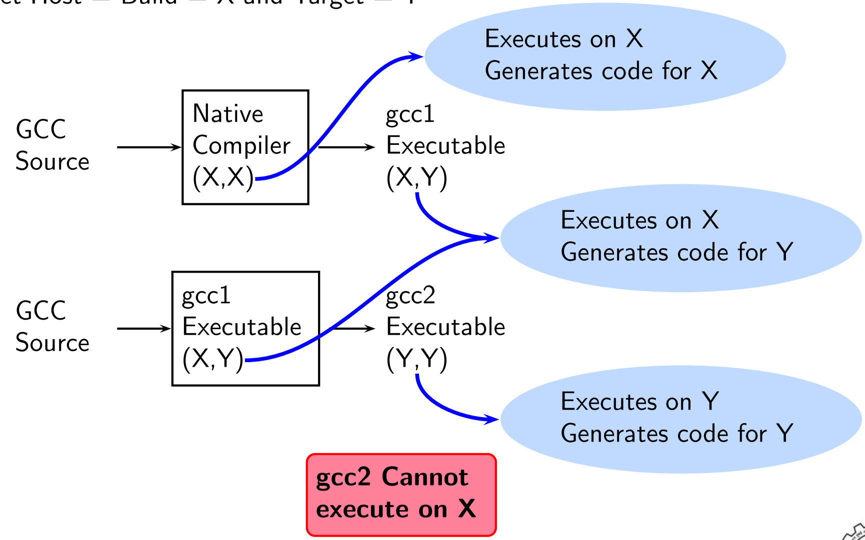
A Cross Build

Let Host = Build = X and Target = Y



A Cross Build

Let Host = Build = X and Target = Y



GCC Code Organization Overview

Logical parts are:

- Build configuration files
- Compiler sources
- Emulation libraries
  `libgcc` $\longrightarrow$ emulate operations not supported on the target
  `real.c` $\longrightarrow$ floating point
- Language Libraries (except C)
- Support software (e.g. garbage collector)

GCC Code Organization Overview

Logical parts are:

- Build configuration files
- Compiler sources
- Emulation libraries
  `libgcc` $\rightarrow$ emulate operations not supported on the target
  `real.c` $\rightarrow$ floating point
- Language Libraries (except C)
- Support software (e.g. garbage collector)

Our conventions

GCC source directory : `$(GCCHOME)`
GCC build directory : `$(GCCBUILD)`
GCC install directory : `$(GCCINSTALL)`
`$(GCCHOME)` $\neq$ `$(GCCBUILD)` $\neq$ `$(GCCBUILD)`

Organization of the Front End Code

- Source language dir: `$(GCCHOME)/<lang dir>`
- Source language dir contains
  - Parsing code
    either hand written or parser generator input
  - Additional AST/Generic nodes, if any
  - Interface to Generic creation

Except for C – which is the "native" language of the compiler

C front end code in: `$(GCCHOME)/gcc`

GCC Backend Organization

- `$(GCCHOME)/gcc/config/<target dir>/`
  Directory containing backend code
- Two main files: `<target>.h` and `<target>.md`,
  e.g. for an `i386` target, we have
  `$(GCCHOME)/gcc/config/i386/i386.md` and
  `$(GCCHOME)/gcc/config/i386/i386.h`
- Usually, also `<target>.c` for additional processing code
  (e.g. `$(GCCHOME)/gcc/config/i386/i386.c`)
- Some additional files

The GCC Build System I

Some Information

- Build-Host-Target systems inferred for native builds
- Specify Target system for cross builds
  Build $\equiv$ Host systems: inferred
- Build-Host-Target systems can be explicitly specified too
- For GCC: A "system" = three entities
  - "cpu"
  - "vendor"
  - "os"

e.g. `sparc-sun-sunos`, `i386-unknown-linux`, `i386-gcc-linux`

The GCC Build System II

Basic GCC Building How To

- `prompt$ cd $GCCBUILD`
- `prompt$ $GCCHOME configure <options>`
  - Specify target: optional for native builds, necessary for others (option `--target=<host-cpu-vendor string>`)
  - Choose source languages (option `--enable-languages=<CSV lang list (c,java)`)
  - Specify the installation directory (option `--prefix=<absolute path of $(GCCBUILD)>`)

  ⇒ `configure` output: customized `Makefile`
- `prompt$ make 2> make.err > make.log`
- `prompt$ make install 2> install.err > install.log`

Adding a New MD

To add a new backend to GCC

- Define a new system name, typically a triple.
  e.g. `spim-gnu-linux`
- Edit `$GCCHOME/config.sub` to recognize the triple
- Edit `$GCCHOME/gcc/config.gcc` to define
  - any backend specific variables
  - any backend specific files
  - `$GCCHOME/gcc/config/<cpu>` is used as the backend directory

  for recognized system names.

Tip

Read comments in `$GCCHOME/config.sub` & `$GCCHOME/gcc/config/<cpu>`.

The GCC Build Process I

GCC builds in two main phases:

- Adapt the compiler source for the specified build/host/target systems
  Consider a cross compiler:
  - Find the target MD in the source tree
  - "Include" MD info into the sources
    (details follow)
- Compile the adapted sources
- NOTE:
  - Incomplete MD specifications ⇒ Unsuccessful build
  - Incorrect MD specification ⇒ Run time failures/crashes
    (either ICE or `SIGSEGV`)

The GCC Build Process

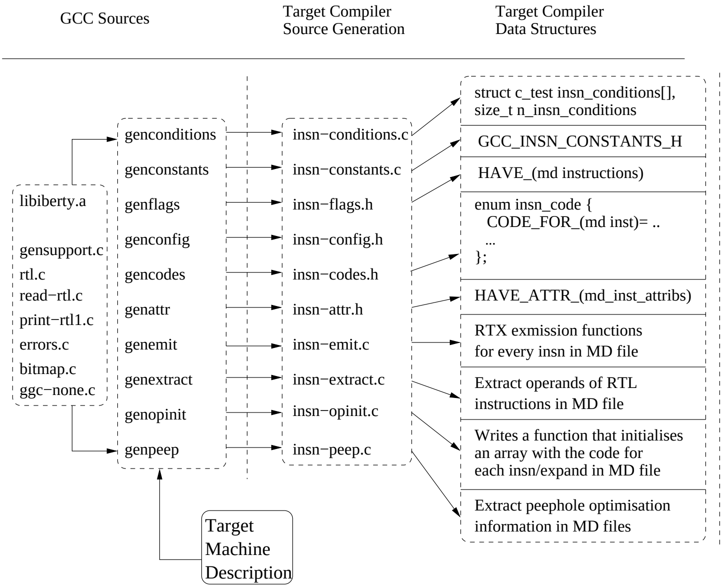
- make first compiles and runs a series of programs that process the target MD
- Typically, the program source file names are prefixed with gen
- The $GCCHOME/gcc/gen*.c programs
  - read the target MD files, and
  - extract info to create & populate the main GCC data structures

Example

Consider genconstants.c:

- <target>.md may define UNSPEC_* constants.
- genconstants.c – reads UNSPEC_* constants
- genconstants.c – generates corresponding #defines
- Collect then into the insn-constants.h
- #include "insn-constants.h" in the main GCC sources

The GCC Build Process



Building GCC – Summary

- Choose the source language: C
  (`--enable-languages=c`)
- Choose installation directory:
  (`--prefix=<absolute path>`)
- Choose the target for non native builds:
  (`--target=sparc-sunos-sun`)
- Run: `configure` with above choices
- Run: `make` to
  - generate target specific part of the compiler
  - build the entire compiler
- Run: `make install` to install the compiler

Tip

Redirect all the outputs:

```
$ make > make.log 2> make.err
```