llvmc2 - New LLVM Compiler Driver

Anton Korobeynikov
asl@math.spbu.ru

Mikhail Glushenkov
foldr@codedgers.com

Outline

1. Motivation
2. Different ways to solve the problem
3. Requirements
4. High-level overview of llvmc2
5. Some low-level details

Compiler Driver

What compiler driver is?

Auxiliary tool doing easy job: execute sequence of used tools to produce output file

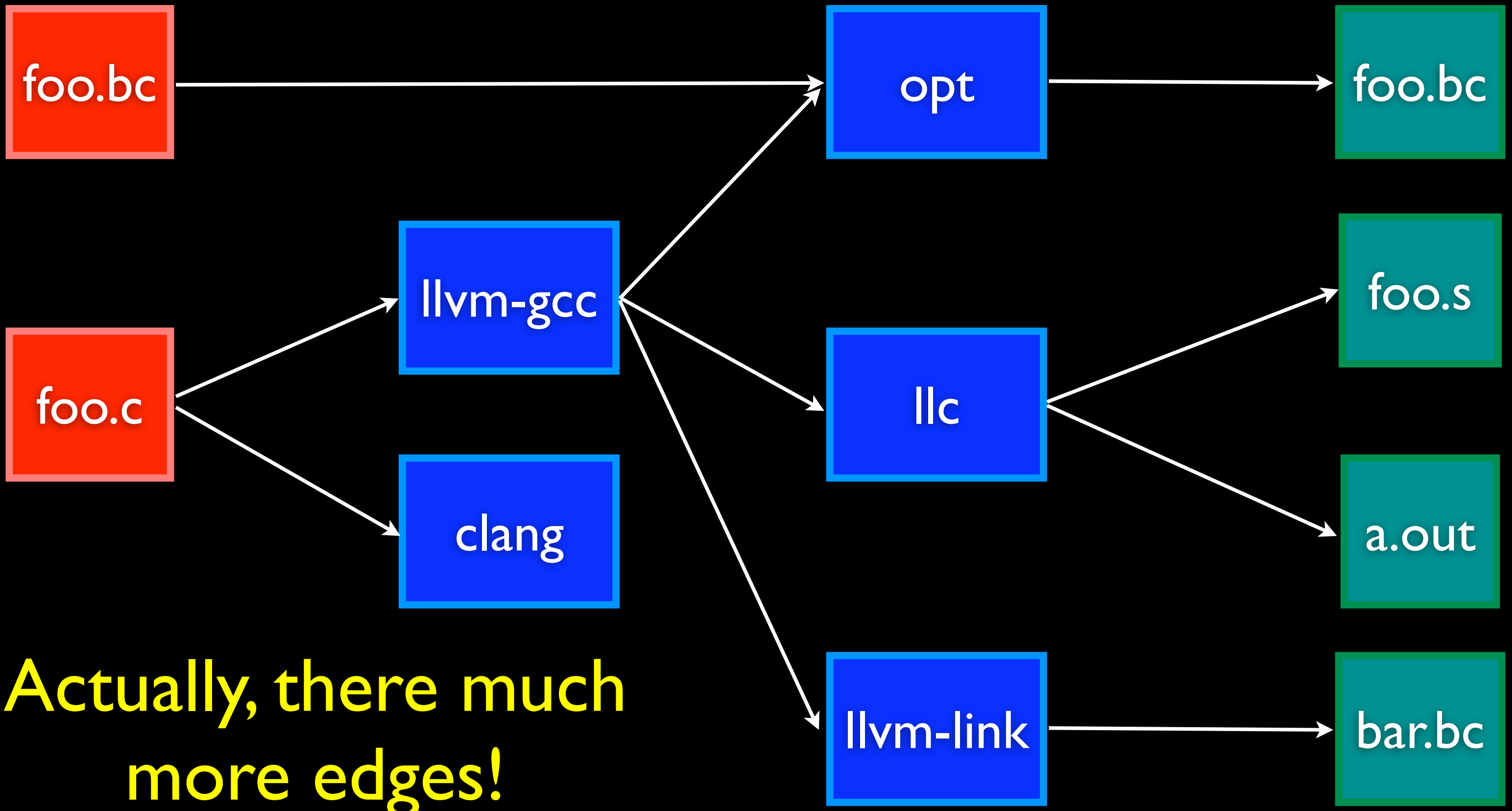
But:

- It should be able to deal with mix of inputs
- It should know about options of all tools and how to dispatch cmdline arguments to them
- And many other small (and not so small) things

Motivation

- LLVM is huge: many tools can be built on top of LLVM libraries
- LLVM is flexible: these tools normally have many options
- Users want all-in-one solution working as a replacement of their favorite compiler / tool out-of-the-box

Small Example



Actually, there much more edges!

Possible solutions

- Hand-written driver:

  Apple gcc's driver-driver

Works fine in its own field, but nowhere else

- Old llvmc

Too weak and restrictive in features

Not suitable for generic compiler driver!

Possible solutions

- gcc specs

In general it works, but:

  - Syntax is little bit ugly
  - You need to make changes in several places at once

- Fully-featured build system

Works, but… really huge overkill

Not suitable in general, but some ideas can be used!

Requirements

- Easy configurable
- No extra dependencies (no perl, python, etc)
- Re-configurable at runtime
- Flexible and universal
- ….

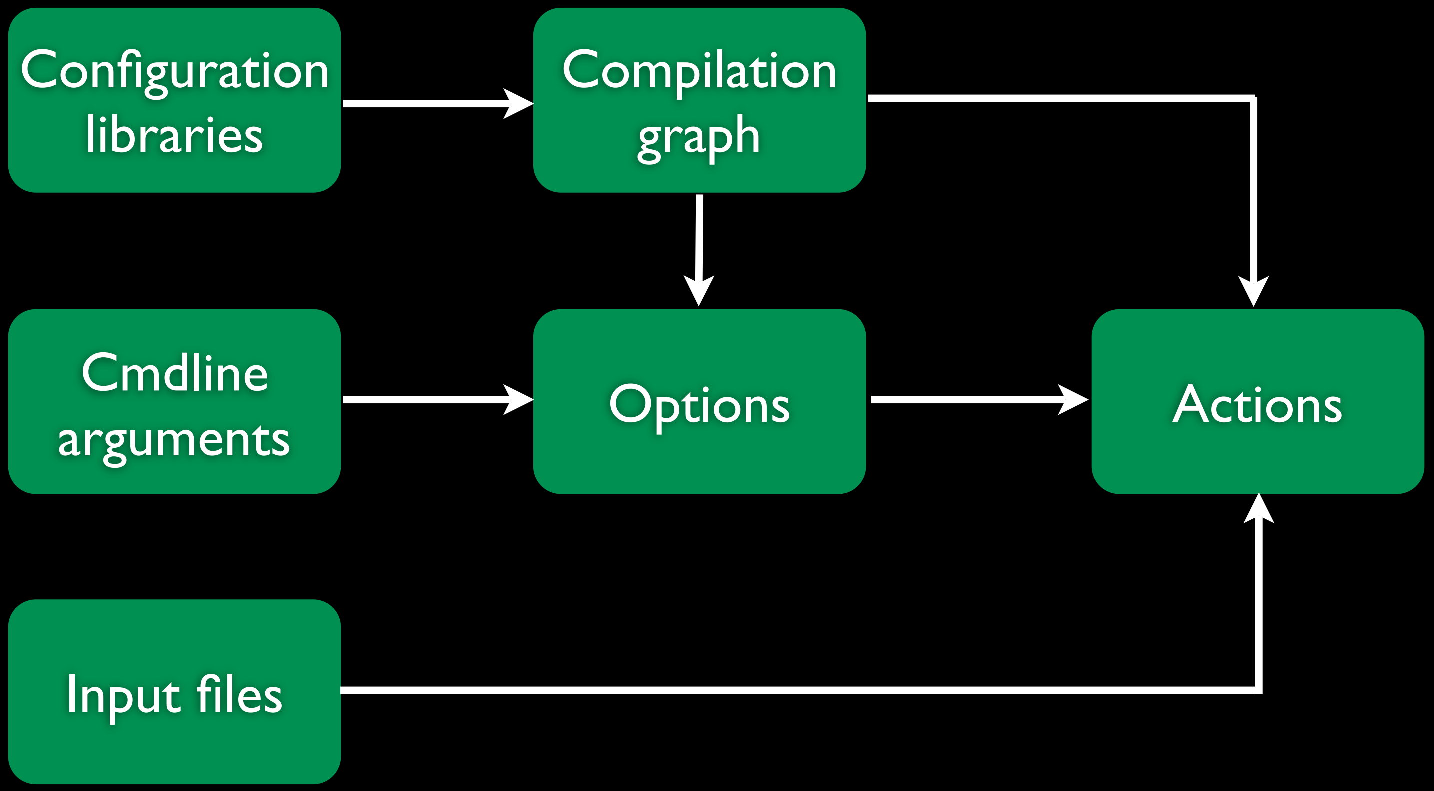
Proposed solution

- Partially inspired by SCons toolkit
- Built as another TableGen backend
- Graph-based approach to describe logic
- Automatic selection of best compilation way
- Driven by set of input files and command line arguments

Graph-based approach

- Different tools define nodes in the transformation graph
- Edges define possible transformation path
- Edges are weighted (weight depends on cmdline, etc.)
- Compilation path with maximal weight is used

High-level overview



N.B.: No configuration libraries at this moment, everything is hardcoded

Options

- Switch option: ‘-time’
- Parameter option:‘-std=c99’
- Parameter option list: ‘-foo=bar -foo=baz’
- Prefix option: ‘-lstdc++’
- Prefix option list: ‘-lm -lpthread’
- Aliases: ‘quiet’ = ‘q’

Options & Actions

Each option has own ‘action’:

```
(prefix_list_option "L", (forward),
                        (help "add a directory to link
path")),
(prefix_list_option "l", (forward),
                        (help "search a library when
linking")),
```

This will make the following supported:

‘llvmc2 -lm -lpthread -Wl,-dead_strip’

Tool

Nodes of the compilations graph describing how exactly the input file will be compiled

```
def llvm_gcc_cpp : Tool<[
    (in_language "c++"),
    (out_language "llvm-assembler"),
    (output_suffix "bc"),
    (cmd_line "llvm-g++ -c $INFILE -o $OUTFILE -emit-llvm"),
    (sink)
]>;
```

‘sink’ will just forward all unused command line options to tool

Tools & Command Lines

One can use hooks to construct command lines:

```
(cmd_line "$CALL(Hook1)/path/to/file -o $CALL(Hook2)")
```

This will call std::string hooks::Hook1() and std::string hooks::Hook2()

Environmental variables can be used in the same manner:

```
(cmd_line "$ENV(VAR1)/path/to/file -o $ENV(VAR2)")
```

Conditional execution is supported as well:

```
(cmd_line
  (case (switch_on "E"),  "llvm-g++ -E -x c $INFILE -o $OUTFILE",
          (default), "llvm-g++ -c -x c $INFILE -o $OUTFILE -emit-llvm"))
```

Tool & Special Stuff

Tools usually do file-by-file transformations . This is not true for linkers. Use join nodes for combining several inputs:

```
def llvm_gcc_linker : Tool<[
    ...
    (cmd_line "llvm-gcc $INFILE -o $OUTFILE"),
    (join),
    (prefix_list_option "L", (forward)),
    (prefix_list_option "l", (forward)),
    (prefix_list_option "Wl", (unpack_values))
]>;
```

N.B.: Currently join nodes should be the last in the compilation chain

Compilation Graph

Used to define compilation chains and glue tools:

```
def CompilationGraph : CompilationGraph<[
    Edge<root, llvm_gcc_c>,
    Edge<root, llvm_gcc_assembler>,
    Edge<llvm_gcc_c, llc>,
    OptionalEdge<llvm_gcc_c, opt, [(switch_on "opt")]>,
    Edge<opt, llc>,
    Edge<llc, llvm_gcc_assembler>,
    Edge<llvm_gcc_assembler, llvm_gcc_linker>
]>;
```

Language Map

- Used to map different input file extensions to input languages
- Tools have input and output languages defined
- One can change the way of compilation with tests on input language

```
def LanguageMap : LanguageMap<
    [LangToSuffixes<"c++", ["cc", "cxx", "cpp", "c++"]>,
     LangToSuffixes<"c", ["c"]>,
     LangToSuffixes<"assembler", ["s"]>,
     LangToSuffixes<"llvm-assembler", ["ll"]> ]>;
```

Option List

Easy way to separate tool-dependent and tool-independent properties of cmdline arguments

```
def Options : OptionList<[
    (switch_option "E", (help "Stop compilation after preprocessing")),
    (alias_option "quiet", "q")
    ...
    ]>;
```

Only options properties are allowed here (‘help’, ‘required’) and option aliases.

Conditional Execution

- case language construction can be used to change compilation flow and alter tool properties
- Tests on different things are available
  - command line options
  - input language
- Look into documentation for full list of tests currently supported

Thank you!

- We really need your suggestions about new and current features of compiler driver
- Detailed documentation can be found at www.llvm.org/docs/CompilerDriver.html
- Examples are in tools/llvmc2 directory