Frontend Option Parsing

`CompilerInvocation` to `-cc1` command line

Daniel Grumberg — LLVM Dev Meeting 2020

Agenda

Modules support and deterministic command lines

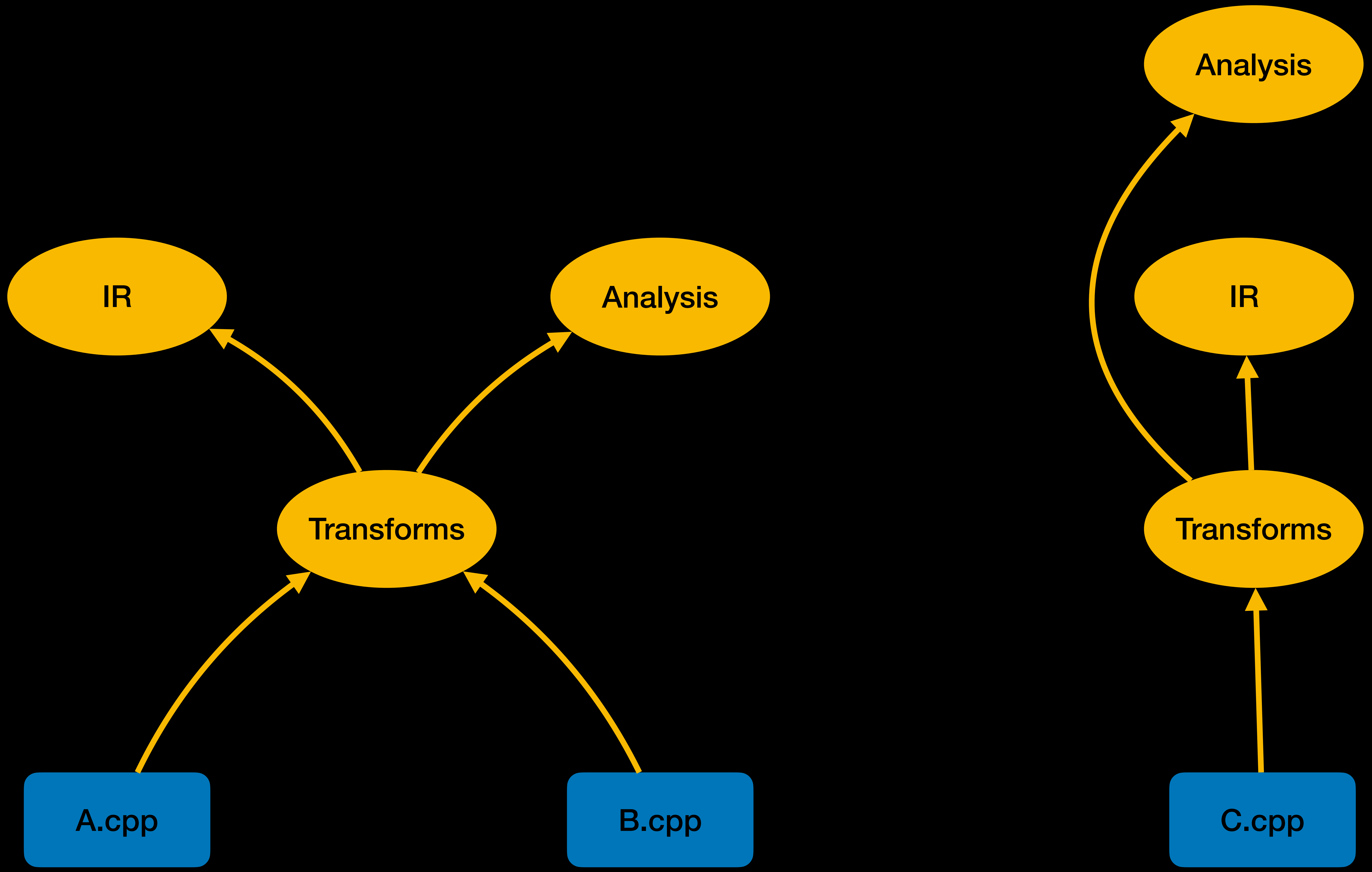
A new option parsing scheme

Generating deterministic command lines

Modules support

The need for deterministic command lines

- Modules replace textual preprocessor includes with an AST import
- We are interested in explicit module builds enabled by clang-scan-deps
- We need to provide the build system with a command line that can build the module


IR
Analysis
Transforms
A.cpp
B.cpp
Analysis
IR
Transforms
C.cpp

Non-deterministic command lines is confusing for the build system

Agenda

Modules support and deterministic command lines

A new option parsing scheme

Generating deterministic command lines

A new option parsing scheme

Convenience TableGen Definitions

Flags	Single Value	Multiple Values	Interdependent options
MarshallingInfoFlag	MarshallingInfoString	MarshallingInfoMultiValueFlag	MarshallingInfoGroup
MarshallingInfoNegativeFlag	MarshallingInfoStringInt		
MarshallingInfoBitfieldFlag	MarshallingInfoEnum		
MarshallingInfoBooleanFlag			

A new option parsing scheme

```
#define OPTION_WITH_MARSHALLING(                                            \
    PREFIX_TYPE, NAME, ID, KIND, GROUP, ALIAS, ALIASARGS, FLAGS, PARAM,     \
    HELPTEXT, METAVAR, VALUES, SPELLING, ALWAYS_EMIT, SHOULD_PARSE, KEYPATH, \
    DEFAULT_VALUE, NORMALIZER, DENORMALIZER, MERGER, EXTRACTOR, TABLE_INDEX) \


  this->KEYPATH = MERGER(this->KEYPATH, DEFAULT_VALUE);                     \
  if (SHOULD_PARSE)                                                         \
    if (auto MaybeValue = NORMALIZER(OPT_##ID, TABLE_INDEX, Args, &Diags))  \
      this->KEYPATH = MERGER(                                               \
          this->KEYPATH, static_cast<decltype(this->KEYPATH)>(*MaybeValue));
```

Agenda

Modules support and deterministic command lines

A new option parsing scheme

Generating deterministic command lines

Generating the command line

```
void CompilerInvocation::generateCC1CommandLine(
    SmallVectorImpl<const char *> &Args, StringAllocator SA) const {


#define OPTION_WITH_MARSHALLING(                                       \
    PREFIX_TYPE, NAME, ID, KIND, GROUP, ALIAS, ALIASARGS, FLAGS, PARAM,   \
    HELPTEXT, METAVAR, VALUES, SPELLING, ALWAYS_EMIT, SHOULD_PARSE, KEYPATH, \
    DEFAULT_VALUE, NORMALIZER, DENORMALIZER, MERGER, EXTRACTOR, TABLE_INDEX) \


if ((FLAGS) & options::CC1Option) {                                    \
    const auto &Extracted = EXTRACTOR(this->KEYPATH);                  \
    if (ALWAYS_EMIT ||                                                 \
        static_cast<decltype(DEFAULT_VALUE)>(Extracted) != DEFAULT_VALUE) \
      DENORMALIZER(Args, SPELLING, SA, Option::KIND##Class, TABLE_INDEX, \
                   Extracted);                                         \
}
```

Generating the command line

Ensuring generating command lines doesn't get broken

- Moved all subfields of `CompilerInvocation` into `.def` files

- Added expensive check in `CompilerInvocation::CreateFromArgs`
    - Generates the command line and parses it again
    - Check that all the fields of the two `CompilerInvocation` instances match

```
#endif
```