Indexing Large, Mixed-Language Codebases

Luke Zarko <zarko@google.com>
The Kythe project aims to establish open data formats and protocols for interoperable developer tools.
Outline

• Introduction
• System structure
• C++ support via Clang
  ○ What does Kythe get?
  ○ What does Kythe propose to give back?
• Future work
I use languages with property $\text{X}$ and I’d like to do $\text{Y}$.
I also use source code generator X, build system Y, repo Z

- protobuf
- thrift
- cap’n proto
- yacc
- antlr
- jni?
- cmake
- gmake
- omake
- mvn
- a bunch of shell scripts
- ant?
- git
- svn
- cvs
- company filer
- local disk
- someone’s :80?
I use tools that support Kythe data
Outline

- Introduction
- **System structure**
- C++ support via Clang
  - What does Kythe get?
  - What does Kythe propose to give back?
- Future work
A Kythe system

cmake

Web browser
A Kythe system

- Extractors pull compilation information from the build system

Diagram:

- cmake
  - compilation database JSON
  - C++ extractor (Clang tool)
  - hermetic build data
  - ...

- Web browser
Hermetic build data

- Contains every dependency the compiler needs for semantic analysis
- Gives files identifiers that can be used to locate them in repositories
- Allows for distribution of analysis tasks
A Kythe system

- Extractors pull compilation information from the build system

- cmake
- Web browser
- C++ extractor (Clang tool)
- compilation database JSON
- hermetic build data
A Kythe system

- Extractors pull compilation information from the build system
- Indexers use this information to construct a persistent graph
Indexer implementation

1. Load hermetic build data into memory with `mapVirtualFile`
2. First pass: recover parent relationships for naming
Nameless decls and shadowed names

- Clang omits parent edges in the AST because it doesn't need them
- As best we can, we want to give stable names to any Decl we see referenced at any point
- We also want to distinguish between shadowed names
- Solution: build a map from AST nodes to (parent, visitation-index)*

```c
void foo() {
    int x;
    x:0:1:0:foo
    { int x; }  
    x:0:2:0:foo
    { int x; }  
}
```
Indexer implementation

1. Load hermetic build data into memory with `mapVirtualFile`
2. First pass: recover parent relationships for naming
Indexer implementation

1. Load hermetic build data into memory with `mapVirtualFile`
2. First pass: recover parent relationships for naming
3. Second pass: notify a `GraphObserver` about abstract program relationships
The Kythe graph

All programs in Kythe are abstracted away to nodes and edges.

<table>
<thead>
<tr>
<th>(some, unique, name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/kythe/node/kind</td>
</tr>
<tr>
<td>/your/own/fact</td>
</tr>
</tbody>
</table>
The Kythe graph

Nodes represent semantic information as well as syntactic information.

<table>
<thead>
<tr>
<th>(some, unique, name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/kythe/node/kind</td>
</tr>
<tr>
<td>/your/own/fact</td>
</tr>
</tbody>
</table>

the class C

“class C” in a particular file

<table>
<thead>
<tr>
<th>(another, unique, name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/kythe/node/kind</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
The Kythe schema

- We provide a base set of nodes and edges
- We also provide rules for naming certain kinds of nodes
- It is extensible: you’re free to use your own node and edge kinds
- “Be conservative in what you send, be liberal in what you accept”
  - some data may be missing
  - there may be more data than you can understand
  - others may produce incorrect data
The schema provides checked examples

```java
// @Enum defines Enumeration
class Enum {
    // @Etor defines Enumerator
    Etor
}
// Enumerator childof Enumeration
```
The GraphObserver is notified about program structure

- The GraphObserver interface sees an abstract view of a program
- There is not a 1:1 mapping between AST nodes and program graph nodes

```
ClassTemplatePartialSpecializationDecl
  Abs
  childof
  Record
```
A Kythe system

- Extractors pull compilation information from the build system
- Indexers use this information to construct a persistent graph

Diagram:
- cmake
- Web browser
- C++ extractor (Clang tool)
- C++ indexer (Clang tool)
- Graph store
- compilation database JSON
- hermetic build data
- Kythe graph nodes and edges

Web browser

Graph store
A Kythe system

- Extractors pull compilation information from the build system
- Indexers use this information to construct a persistent graph
- Services use the graph to answer queries
  - code browsing
  - code review
  - documentation generation
This design is known to scale

- Small dataset (Chromium)
  - ~22,600 C++ compilations
  - ~31G of serving data
- Internal code search is much larger
  - 100 million lines of code
- Other internal tools make use of build data for analysis
Outline

- Introduction
- Rough system structure
- **C++ support via Clang**
  - What does Kythe get?
  - What does Kythe propose to give back?
- Future work
Clang made C++ tooling possible

- A tooling-friendly compiler leads to an ecosystem of software tools
  - ASan, TSan, MSan
  - clang-format, clang-tidy
  - Doxygen libclang integration

- Clang’s code is eminently hackable
  - The interface to the typed AST is clean
  - The preprocessor is easy to tool as well
Clang has excellent template support

template <typename T> class C
{ typename T::Foo foo; };  // ClassTemplateDecl (of CXXRecordDecl)

template <typename S> class C<S*> 
{ typename S::Bar bar; };  // ClassTemplatePartialSpecializationDecl

template <> class C<int> { };  // ClassTemplateSpecializationDecl

C<X> CX;
C<X*> CPX;
C<int> CI;  // implicit ClassTemplateSpecializationDecl
Clang has excellent template support

template <typename T> class C
{
    typename T::Foo foo;
};

template <typename S> class C<S*>
{   typename S::Bar bar;
};

C<X> CX;
C<X*> CPX;
C<int> CI,

"template <X*> class C"

.getTemplateArgs
=> { X* }  

"template <X=T> class C"

.getTemplateInstantiationArgs
=> { X }  

"template <X=S> class C<X*>"
Clang makes macros manageable

#define M1(a,b) ((a) + (b))
int f() {
    int x = 0, y = 1;
    return M1(x, y);
}

Clang makes macros manageable

#define M1(a,b) ((a) + (b))
int f() {
    int x = 0, y = 1;
    return M1(x, y);
}
Clang supports other compilers’ extensions: GCC

- We want to index real world code!
- Just some of the GCC extensions clang supports:
  - indirect-goto (`goto *bar;`)
  - address-of-label (`void *bar = &&foo;`)
  - statement-expression
    (`string s("?"); ({for(;;); s;}).size();`)
  - conditional expression without middle operand (`f() ? : g()`)
  - case labels with ranges (`case 'A' ... 'Z':`)
  - ranges in array initializers
    `int a[] = { [0 ... 9] = 1, [10 ... 99] = 2, [100] = 3 };`
Clang can build extension-heavy software

- Building the Linux kernel works (modulo some patches: http://llvm.linuxfoundation.org/index.php/Main_Page)
- Hairiest GCC “feature” unsupported: variable length arrays in structs
  ```c
  struct {
    struct shash_desc_desc shash;
    char ctx[crypto_shash_descsize(tfm)];
  } desc;
  ```
- Support for MSVC extensions (and ABI…) is developing too; some success with Chromium on Windows (https://code.google.com/p/chromium/wiki/Clang)
Kythe adds to Clang’s tooling support

- Persistence for abstract program data: records, not CXXRecordDecl's.
- Hermetic storage of compilation units
- Unambiguous naming for more program entities
- Abstract AST traversal
C++ is a first-class citizen

- The Kythe schema is intended to support all of C++14 (templates, (generic) lambdas, auto, …)
- We expect support for Concepts Lite will not be difficult
- To get this into Clang:
  - Nothing Kythe-specific goes into the LLVM tree
  - Just a library in clang/tools/extra that calls appropriate members on an abstract GraphObserver
  - The Kythe indexer is a particular implementation of GraphObserver
Outline

- Introduction
- System structure
- C++ support via Clang
  - What does Kythe get?
  - What does Kythe propose to give back?
- Future work
Things left to do

- UI/IDE integration
- Support for other languages
  - Including one or two that are supported by Clang already
- Other analyses that work over or contribute to the graph
  - Use Kythe information as sparse data to drive whole-project analysis
- Adding more build information (eg, who links to whom)
- Quick incremental updates
Summary

- The open Kythe data format enables interoperable tooling
- The Kythe pipeline is designed to scale
- C++ support is possible thanks to the work done on Clang tooling
- Simpler languages (Go, Java) aren’t necessarily easier to tool
- The code we will propose to upstream does not depend on Kythe
- There are lots of opportunities for community development
Mailing list

https://groups.google.com/forum/#!forum/kythe-early-interest