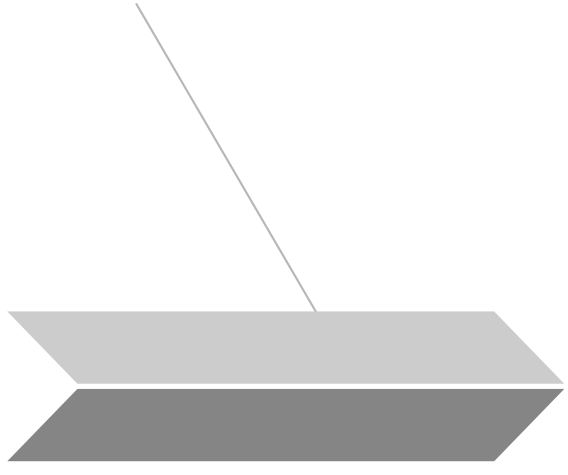

Pets, Cattle and
Automatic Operations
with Code
Svyatoslav Feldsherov
2024

Hi, I am Slava!

My story TL/DR

My story TL/DR

Tons of code

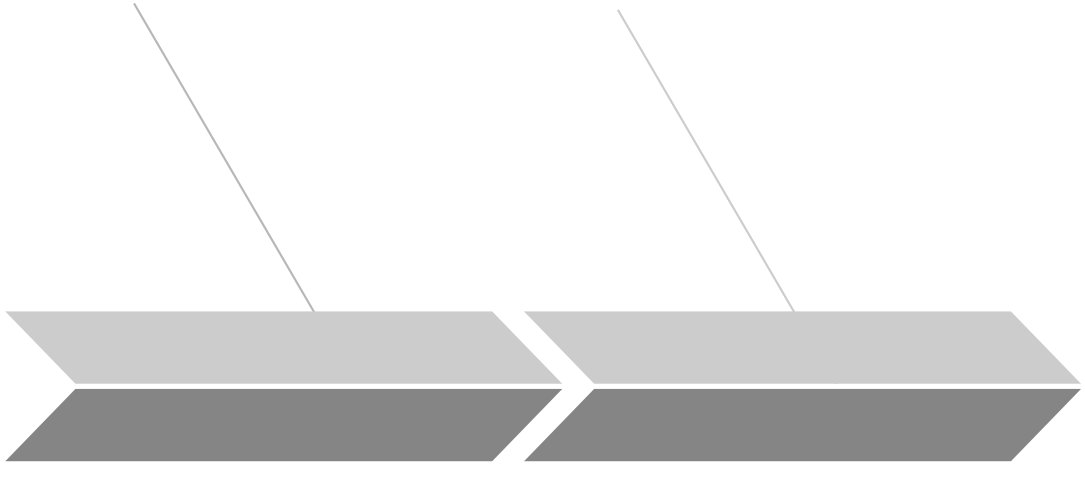


Yandex Ads Infra

My story TL/DR

Tons of code

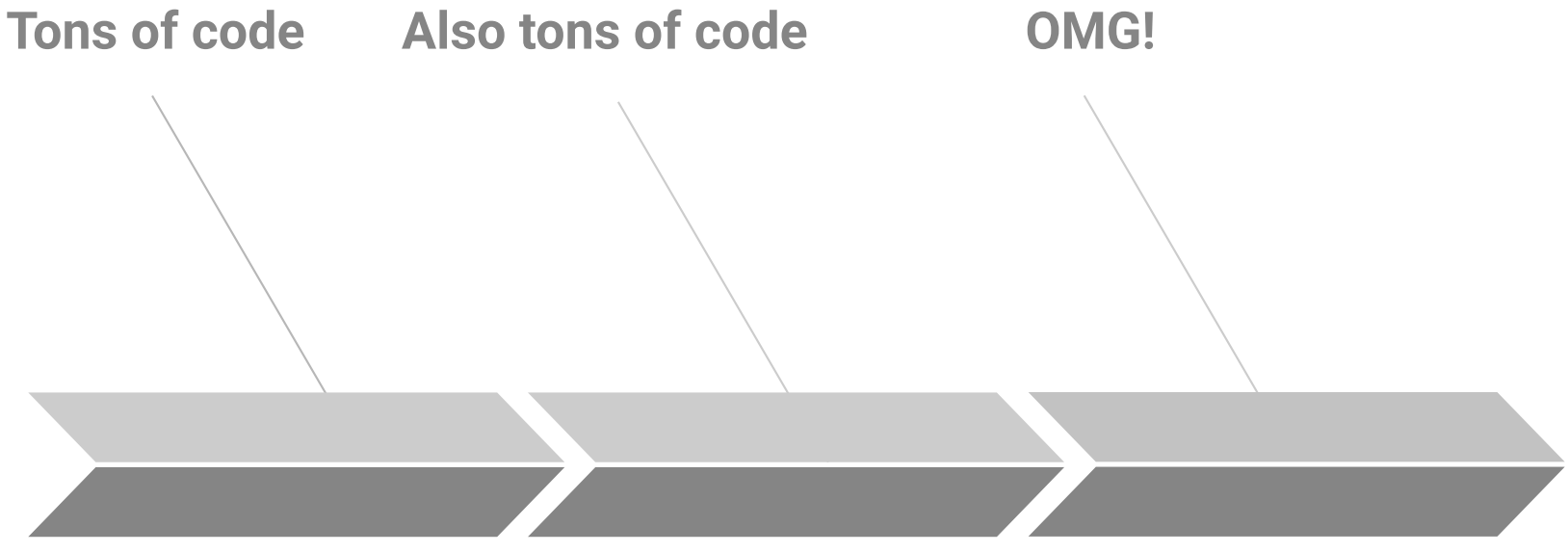
Also tons of code



Yandex Ads Infra

Yandex Search Infra

My story TL/DR



New cpp feature arrived!

How to migrate codebase?

Modify widely used API

And just tasks....

When you practice regexps over code

Are huge projects only a corporation’s disease?

Wave me if your project larger 100k loc

How to make life simpler?

Let’s treat code as cattle!

Automate what?

Automate what?

Find

Automate what?

Find

- Find all function declarations with const std::string& argument

Automate what?

Find

- Find all function declarations with const std::string& argument

Fix and enforce

Automate what?

Find

- Find all function declarations with const std::string& argument

Fix and enforce

- Replace std::enable_if with concept's requires

Automate what?

Find

- Find all function declarations with const std::string& argument

Fix and enforce

- Replace std::enable_if with concept's requires

Generate

```
1 enum class Color {
2   GREEN  = 0,
3   YELLOW = 1,
4   RED    = 2,
5 };
```

```
std::string ColorToString(Color c) {
  switch (c) {
    case Color::GREEN:
        return "GREEN";
    case Color::YELLOW:
        return "YELLOW";
    case Color::RED:
        return "RED";
  }
}
```

```cpp
std::string ColorToString(Color c) {
  switch (c) {
    case Color::GREEN:
        return "GREEN";
    case Color::YELLOW:
        return "YELLOW";
    case Color::RED:
        return "RED";
  }
}
```

```
std::string ColorToString(Color c) {
  switch (c) {
    case Color::GREEN:
        return "GREEN";
    case Color::YELLOW:
        return "YELLOW";
    case Color::RED:
        return "RED";
  }
}
```

```
std::string ColorToString(Color c) {
  switch (c) {
    case Color::GREEN:
        return "GREEN";
    case Color::YELLOW:
        return "YELLOW";
    case Color::RED:
        return "RED";
  }
}
```

Automate what?

Find

Fix and enforce

Generate

Theory!

There are grep, sed and company! Isn’t it enough?

std::condition_variable::wait

```
 1 class Worker {
 2
 3   void work() {
 4     std::unique_lock lock{mt_};
 5     if (!ready_) {
 6       cv_.wait(lock);
 7     }
 8     ...
 9   }
10
11   std::mutex mt_;
12   bool ready_;
13   std::condition_variable cv_;
14 };
```

std::condition_variable::wait

```
1  class Worker {
2
3    void work() {
4      std::unique_lock lock{mt_};
5      if (!ready_) { // <- spurious wake up issue
6        cv_.wait(lock);
7      }
8      ...
9    }
10
11   std::mutex mt_;
12   bool ready_;
13   std::condition_variable cv_;
14 };
```

std::condition_variable::wait

```
class Worker {

  void work() {
    std::unique_lock lock{mt_};
    cv_.wait(lock, [this](){
      return ready_;
    });
    ...
  }

  std::mutex mt_;
  bool ready_;
  std::condition_variable cv_;
};
```

Let’s try something like this…

```
grep -rHn --include "*.cc" --include "*.h" -E 'wait\([^,]+\)'
```

You will find function in C-library

```
void wait(int sec) {
  ...
}
```

... and some debug output

```
int far_far_away() {
  ...
  std::cout << "useful debug about wait(delay)" << std::endl;
}
```

… and formatting will make life hard

```
long_condvar_name.wait(
  long_mutex_name
);
```

And depression will come

We are missing structure!

```
1 std::condition_variable condvar_name;
2
3 condvar_name.wait(
4   long_mutex_name
5 );
```

We are missing structure!

```
1 std::condition_variable condvar_name;
2
3 condvar_name.wait(
4   long_mutex_name
5 );
```

We are missing structure!

```
1 std::condition_variable condvar_name;
2
3 condvar_name.wait(
4   long_mutex_name
5 );
```

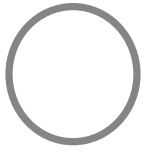
We are missing structure!

```
1 std::condition_variable condvar_name;
2
3 condvar_name.wait(
4   long_mutex_name
5 );
```

Ok, but where to find this structure?

Compilation process

Compilation process

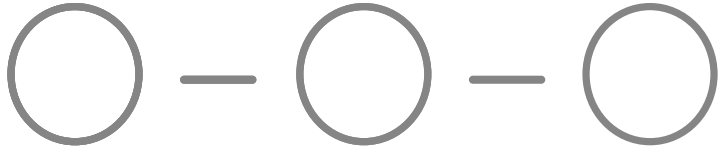


Source

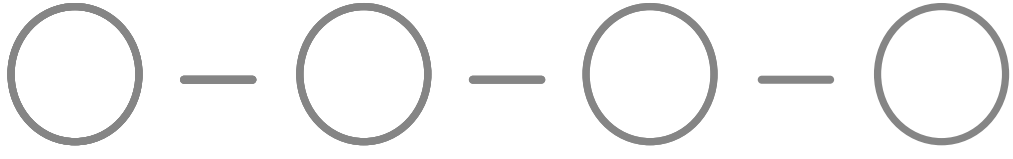
Compilation process

Source Preprocessing

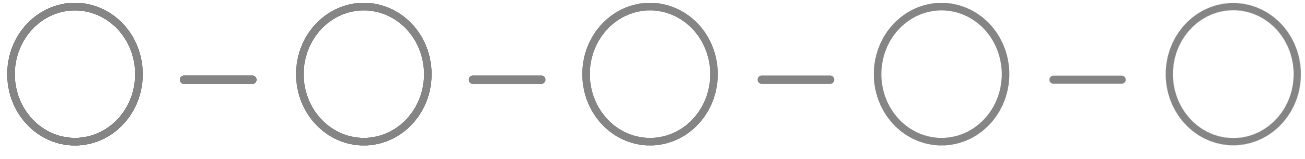
Compilation process



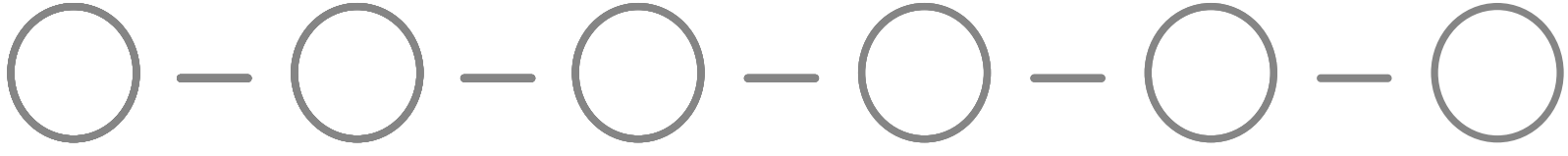
Compilation process



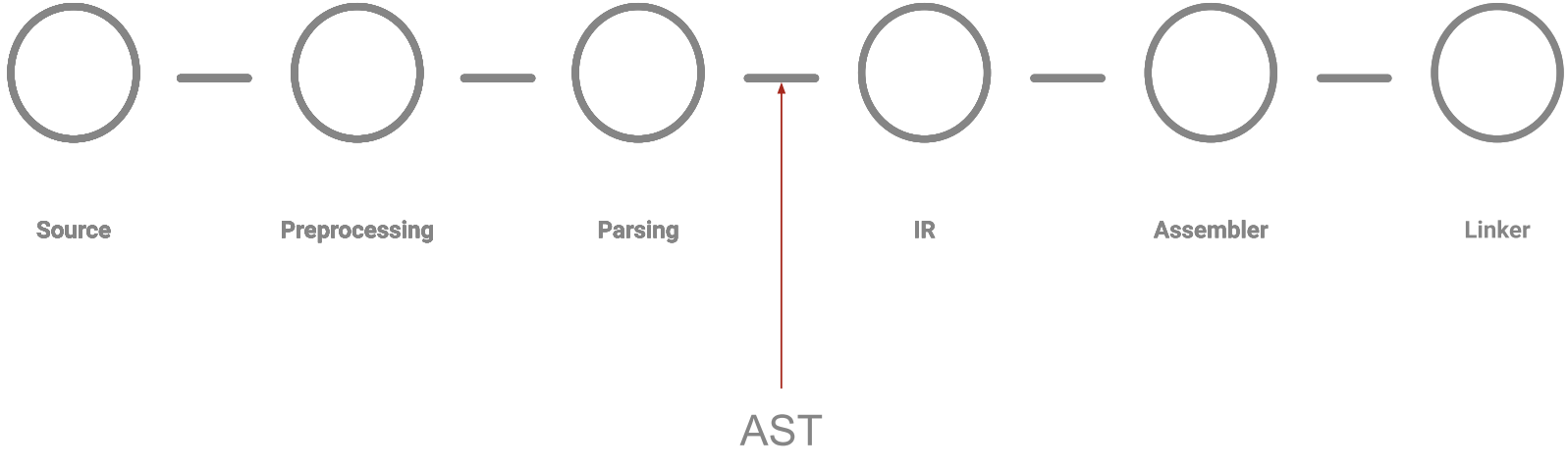
Compilation process



Compilation process



Compilation process



Link to demo

Ok, but how to build tools with it?

Ok, but how to build tools with it?

clang libtooling

Ok, but how to build tools with it?

clang libtooling
clang matchers

Ok, but how to build tools with it?

clang libtooling
clang matchers
clang transform

Ok, but how to build tools with it?

clang libtooling
clang matchers
clang transform
python bindings

Find

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Matcher for wait problem

```
1  m callExpr(
2    callee(
3      cxxMethodDecl(
4        allOf(
5          hasName("wait"),
6          ofClass(hasName("std::condition_variable")))
7        )
8    ),
9    argumentCountIs(1)
10 )
```

[0] Matchers referece

Lost exceptions

```
1 try {
2   do_something();
3 } catch (...) {
4   eat_exception_and_smile();
5 }
```

And the matcher...

```
1 m cxxCatchStmt(
2   unless(
3     hasDescendant(
4         cxxThrowExpr()
5     )
6   )
7 )
```

And the matcher...

```
1 m cxxCatchStmt(
2   unless(
3     hasDescendant(
4         cxxThrowExpr()
5     )
6   )
7 )
```

And the matcher...

```
1 m cxxCatchStmt(
2   unless(
3     hasDescendant(
4         cxxThrowExpr()
5     )
6   )
7 )
```

And the matcher...

```
1 m cxxCatchStmt(
2   unless(
3     hasDescendant(
4         cxxThrowExpr()
5     )
6   )
7 )
```

Won’t match

```
1 try {
2   do_something();
3 } catch (...) {
4   if (false) {
5     throw;
6   }
7 }
```

Will match

```
1 try {
2   do_something();
3 } catch (...) {
4   AlwaysThrow();
5 }
```

Link to demo

clang-query from the terminal example

- [0] Github with examples
- [1] Firefox docs on clang-query

Conclusion

Conclusion

Matchers is a declarative language to query AST

Conclusion

Matchers is a declarative language to query AST

Use clang-query to debug matchers on your code

Conclusion

Matchers is a declarative language to query AST
Use clang-query to debug matchers on your code
Matchers are almost enough for enforcing

Fix and enforce

clang-transform

Pointer to reference argument

```
1 void AwesomeFunc(SomeObject* object);
2 void AwesomeFunc(SomeObject& object);
3
4 int main() {
5   SomeObject object;
6   AwesomeFunc(&object);
7 }
```

Matcher

```
m callExpr(
    callee(
      functionDecl(
        hasName("AwesomeFunc")
      )
    ),
    hasArgument(
      0, expr(
          hasType(pointerType())
      ).bind("argument")
    )
  )
```

Matcher

```
1  m callExpr(
2      callee(
3        functionDecl(
4          hasName("AwesomeFunc")
5        )
6      ),
7      hasArgument(
8        0, expr(
9            hasType(pointerType())
10       ).bind("argument")
11     )
12   )
```

Matcher

```
1  m callExpr(
2      callee(
3        functionDecl(
4          hasName("AwesomeFunc")
5        )
6      ),
7      hasArgument(
8        0, expr(
9            hasType(pointerType())
10       ).bind("argument")
11     )
12   )
```

Matcher

```
1  m callExpr(
2      callee(
3        functionDecl(
4          hasName("AwesomeFunc")
5        )
6      ),
7      hasArgument(
8        0, expr(
9            hasType(pointerType())
10       ).bind("argument")
11     )
12   )
```

Transformation

```
changeTo(
  node("argument"),
  transformer::maybeDeref("argument")
)
```

- [0] Clang transform tutorial

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

More transformations

```
1 insertBefore(node("id"), cat("some text"))
2 addInclude(node("id"), "myaweomelib.h")
3 changeTo(node("argument"), cat("new_name"));
4 changeTo(node("id"), access(node("id"), "myFieldName"))
```

Interesting part of tool with clang-transform

```
const std::string ArgId = "argument";
auto Rule = makeRule(
    callExpr(callee(functionDecl(hasName("AwesomeFunc"))),
             hasArgument(0,
               expr(hasType(pointerType())).bind(ArgId))),
    changeTo(node(ArgId), transformer::maybeDeref(ArgId)));

Transformer TransformerInstance(
  Rule,
  [&AllChanges](
      Expected<llvm::MutableArrayRef<AtomicChange>> Changes
    ) {
    ...
});
```

Interesting part of tool with clang-transform

```
const std::string ArgId = "argument";
auto Rule = makeRule(
    callExpr(callee(functionDecl(hasName("AwesomeFunc"))),
             hasArgument(0,
               expr(hasType(pointerType())).bind(ArgId))),
    changeTo(node(ArgId), transformer::maybeDeref(ArgId)));

Transformer TransformerInstance(
  Rule,
  [&AllChanges](
      Expected<llvm::MutableArrayRef<AtomicChange>> Changes
    ) {
    ...
});
```

Interesting part of tool with clang-transform

```
1  const std::string ArgId = "argument";
2  auto Rule = makeRule(
3      callExpr(callee(functionDecl(hasName("AwesomeFunc"))),
4               hasArgument(0,
5                 expr(hasType(pointerType())).bind(ArgId))),
6      changeTo(node(ArgId), transformer::maybeDeref(ArgId)));
7
8  Transformer TransformerInstance(
9    Rule,
10   [&AllChanges](
11       Expected<llvm::MutableArrayRef<AtomicChange>> Changes
12     ) {
13     ...
14 });
```

Interesting part of tool with clang-transform

```
const std::string ArgId = "argument";
auto Rule = makeRule(
    callExpr(callee(functionDecl(hasName("AwesomeFunc"))),
             hasArgument(0,
               expr(hasType(pointerType())).bind(ArgId))),
    changeTo(node(ArgId), transformer::maybeDeref(ArgId)));

Transformer TransformerInstance(
  Rule,
  [&AllChanges](
      Expected<llvm::MutableArrayRef<AtomicChange>> Changes
    ) {
    ...
});
```

Let’s make it live!

- [0] Github with example

Wait, but... How to enforce?

Clang tidy!

- [0] absl check example with clang tidy

Interesting part of tool with clang-transform

```
1  class YourNewTidyCheck :
2          public utils::TransformerClangTidyCheck {
3  public:
4    YourNewTidyCheck(
5      StringRef Name,
6      ClangTidyContext *Context
7    ) : utils::TransformerClangTidyCheck(
8          MakeRule(),
9          Name,
10         Context) {
11   }
12 };
```

Interesting part of tool with clang-transform

```
1  class YourNewTidyCheck :
2          public utils::TransformerClangTidyCheck {
3  public:
4    YourNewTidyCheck(
5      StringRef Name,
6      ClangTidyContext *Context
7    ) : utils::TransformerClangTidyCheck(
8          MakeRule(),
9          Name,
10         Context) {
11   }
12 };
```

Conclusion

Conclusion

clang-transform is like "search and replace" powered by clang matchers

Conclusion

clang-transform is like "search and replace" powered by clang matchers

clang-tidy support out of the box

Conclusion

clang-transform is like "search and replace" powered by clang matchers
clang-tidy support out of the box
relatively hard to build your first tool

Conclusion

clang-transform is like "search and replace" powered by clang matchers
clang-tidy support out of the box
relatively hard to build your first tool
no way to upstream a really custom tool

Conclusion

clang-transform is like "search and replace" powered by clang matchers
clang-tidy support out of the box
relatively hard to build your first tool
no way to upstream a really custom tool
check clang-tools-extra before writing new :)

Generate

Tons of span objects

```
1 Span some_event("Long explanation what happened.");
```

Target is something like this

```
1  enum class Events {
2    ...
3    kSomeEvent = 179,
4    ...
5  };
6
7  std::vector<std::string_view> s{
8    ...,
9    "Long explanation what happened.",
10   ...
11 };
12
13 Span some_event(Events::kSomeEvent);
```

Target is something like this

```
enum class Events {
  ...
  kSomeEvent = 179,
  ...
};

std::vector<std::string_view> s{
  ...,
  "Long explanation what happened.",
  ...
};

Span some_event(Events::kSomeEvent);
```

Target is something like this

```
enum class Events {
  ...
  kSomeEvent = 179,
  ...
};

std::vector<std::string_view> s{
  ...,
  "Long explanation what happened.",
  ...
};

Span some_event(Events::kSomeEvent);
```

Target is something like this

```
enum class Events {
  ...
  kSomeEvent = 179,
  ...
};

std::vector<std::string_view> s{
  ...,
  "Long explanation what happened.",
  ...
};

Span some_event(Events::kSomeEvent);
```

Target is something like this

```
1  enum class Events {
2    ...
3    kSomeEvent = 179,
4    ...
5  };
6
7  std::vector<std::string_view> s{
8    ...,
9    "Long explanation what happened.",
10   ...
11 };
12
13 Span some_event(Events::kSomeEvent);
```

Python libclang

[0] libclang PyPi

```python
def find_all_spans(cursor: Cursor, result: list):
  if cursor.kind == CursorKind.VAR_DECL:
      type_decl = cursor.type.get_declaration()
      if type_decl.displayname != "Span":
          return

      var_name = cursor.displayname
      message_description = find_string_literal(cursor)
      result.append(...)
      return

  for c in cursor.get_children():
      find_all_spans(c, result)
```

[0] Github example

```python
def find_all_spans(cursor: Cursor, result: list):
  if cursor.kind == CursorKind.VAR_DECL:
      type_decl = cursor.type.get_declaration()
      if type_decl.displayname != "Span":
          return

      var_name = cursor.displayname
      message_description = find_string_literal(cursor)
      result.append(...)
      return

  for c in cursor.get_children():
      find_all_spans(c, result)
```

[0] Github example

```python
def find_all_spans(cursor: Cursor, result: list):
    if cursor.kind == CursorKind.VAR_DECL:
        type_decl = cursor.type.get_declaration()
        if type_decl.displayname != "Span":
            return

        var_name = cursor.displayname
        message_description = find_string_literal(cursor)
        result.append(...)
        return

    for c in cursor.get_children():
        find_all_spans(c, result)
```

[0] Github example

```python
def find_all_spans(cursor: Cursor, result: list):
  if cursor.kind == CursorKind.VAR_DECL:
      type_decl = cursor.type.get_declaration()
      if type_decl.displayname != "Span":
          return

      var_name = cursor.displayname
      message_description = find_string_literal(cursor)
      result.append(...)
      return

  for c in cursor.get_children():
      find_all_spans(c, result)
```

[0] Github example

```python
def find_all_spans(cursor: Cursor, result: list):
    if cursor.kind == CursorKind.VAR_DECL:
        type_decl = cursor.type.get_declaration()
        if type_decl.displayname != "Span":
            return

        var_name = cursor.displayname
        message_description = find_string_literal(cursor)
        result.append(...)
        return

    for c in cursor.get_children():
        find_all_spans(c, result)
```

[0] Github example

Let’s go live again!

Conclusion

Conclusion

libclang is awesome for fast hacking

Conclusion

libclang is awesome for fast hacking
if you need more, then custom tool is your choice

Talk message in one slide

Talk message in one slide

Projects became huge

Talk message in one slide

Projects became huge

There are repetitive tasks with code

Talk message in one slide

Projects became huge

There are repetitive tasks with code

These tasks are far beyond formatting and linting

Talk message in one slide

Projects became huge
There are repetitive tasks with code
These tasks are far beyond formatting and linting
Clang tooling allows to delegate them to machines

Talk message in one slide

Projects became huge
There are repetitive tasks with code
These tasks are far beyond formatting and linting
Clang tooling allows to delegate them to machines
Use this opportunity

QUESTIONS?

- [0] Tg: @feldsherov
- [1] Link: svyat
- [2] Mail: svyat@feldsherov.name