```
domjandaniel@eurollvm-2025:~$ berlin
berlin: symbol lookup error: libeurollvm.so.2025: undefined symbol:
_Z11abiBreakingChange
domjandaniel@eurollvm-2025:~$
```

berlin: symbol lookup er

_Z11abiBreakingChange

domjandanie1@eurol1vm-20

mangled_name_reproducer.cpp

```cpp
#include <complex>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double);
```

mytalk.h

```h
#ifndef MYTALK_H
#define MYTALK_H

#include <complex>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double);

#endif
```

mytalk.cpp

```cpp
#include "mytalk.h"

#include <iostream>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double) {
  std::cout << "Hello EuroLLVM!\n";
}
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$
```

~$ berlin

: libeurollvm.so.25: u

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$
```

```
domjandanie@eurollvm-20
berlin: symbol lookup er
_Z11abiBreakingChange
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$ berlin
Hello EuroLLVM!
domjandaniel@eurollvm-2025:~$
```

mytalk.h

```
#ifndef MYTALK_H
#define MYTALK_H

#include <complex>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double);

#endif
```

mytalk.h

```
#ifndef MYTALK_H
#define MYTALK_H

#include <complex>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double, bool);

#endif
```

mytalk.h

```
#ifndef MYTALK_H
#define MYTALK_H

#include <complex>

void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double, bool = false);

#endif
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$ berlin
Hello EuroLLVM!
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$ berlin
Hello EuroLLVM!
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$ berlin
Hello EuroLLVM!
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ berlin
berlin: symbol lookup error: libeurollvm.so.2025: undefined symbol:
_Z11abiBreakingChange
domjandaniel@eurollvm-2025:~$
```

Shared Library

Shared Library

libeurollvm.so.2025

```
.text:
  _Z8eurollvmv:
    sub     rsp, 0x18
    ...
    call    _Z11abiBreakingChange@PLT
    add     rsp, 0x18
    ret

.dynamic:
  Shared library: [libmytalk.so]
```

Shared Library

Only referenced in applications

libeurollvm.so.2025

```
.text:
  _Z8eurollvmv:
    sub     rsp, 0x18
    ...
    call    _Z11abiBreakingChange@PLT
    add     rsp, 0x18
    ret

.dynamic:
  Shared library: [libmytalk.so]
```

Shared Library

Only referenced in applications

Loaded in runtime

libeurollvm.so.2025

```
.text:
  _Z8eurollvmv:
    sub     rsp, 0x18
    ...
    call    _Z11abiBreakingChange@PLT
    add     rsp, 0x18
    ret

.dynamic:
  Shared library: [libmytalk.so]
```

Shared Library

Only referenced in applications

Loaded in runtime

Easy to break compatibility

libeurollvm.so.2025

```
.text:
  _Z8eurollvmv:
    sub     rsp, 0x18
    ...
    call    _Z11abiBreakingChange@PLT
    add     rsp, 0x18
    ret

.dynamic:
  Shared library: [libmytalk.so]
```

Source Compatibility

Refers to the ability of existing consumers of an API to recompile against a newer version without any source changes.

https://learn.microsoft.com/en-us/dotnet/core/compatibility/categories#source-compatibility

Binary Compatibility

Refers to the ability of a consumer of an API to use the API on a newer version without recompilation.

https://learn.microsoft.com/en-us/dotnet/core/compatibility/categories#binary-compatibility

Application Programming Interface

	DEF	CALL
OLD	`void abiBreaking(...);`	`abiBreaking(...);`
NEW	`void abiBreaking(..., bool = false);`	`abiBreaking(...);`

Application Programming Interface

```
        DEF                                  CALL
OLD     void abiBreaking(...);               abiBreaking(...);
NEW     void abiBreaking(..., bool = false); abiBreaking(...);
```

Application Binary Interface

```
        DEF                                  CALL
OLD     _Z11abiBreakingChange:               call   _Z11abiBreakingChange
NEW     _Z11abiBreakingChangeb:              call   _Z11abiBreakingChangeb
```

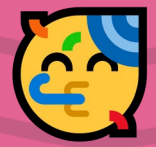
```
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ ld eurollvm.o -shared -o libeurollvm.so.2025 -lmytalk
domjandaniel@eurollvm-2025:~$ ld berlin.o -l:libeurollvm.so.2025 -o berlin
domjandaniel@eurollvm-2025:~$ berlin
Hello EuroLLVM!
domjandaniel@eurollvm-2025:~$ clang++ mytalk.cpp -fpic -shared -o libmytalk.so
domjandaniel@eurollvm-2025:~$ berlin
berlin: symbol lookup error: libeurollvm.so.2025: undefined symbol:
_Z11abiBreakingChange
domjandaniel@eurollvm-2025:~$
```

But there are already multiple tools detecting them...

Analyse the content of the binary…

ELF, PE, Mach-O 😒

Multiple platforms use DWARF...

PDB 

Analyze the source code...

No such tool for C++ 😢

LibTooling 🤔

LibASTMatchers 💭

old/header.h

```
class Class {
  void privateMethod();

public:
  void publicMethod() {
    privateMethod();
  }
};
```

new/header.h

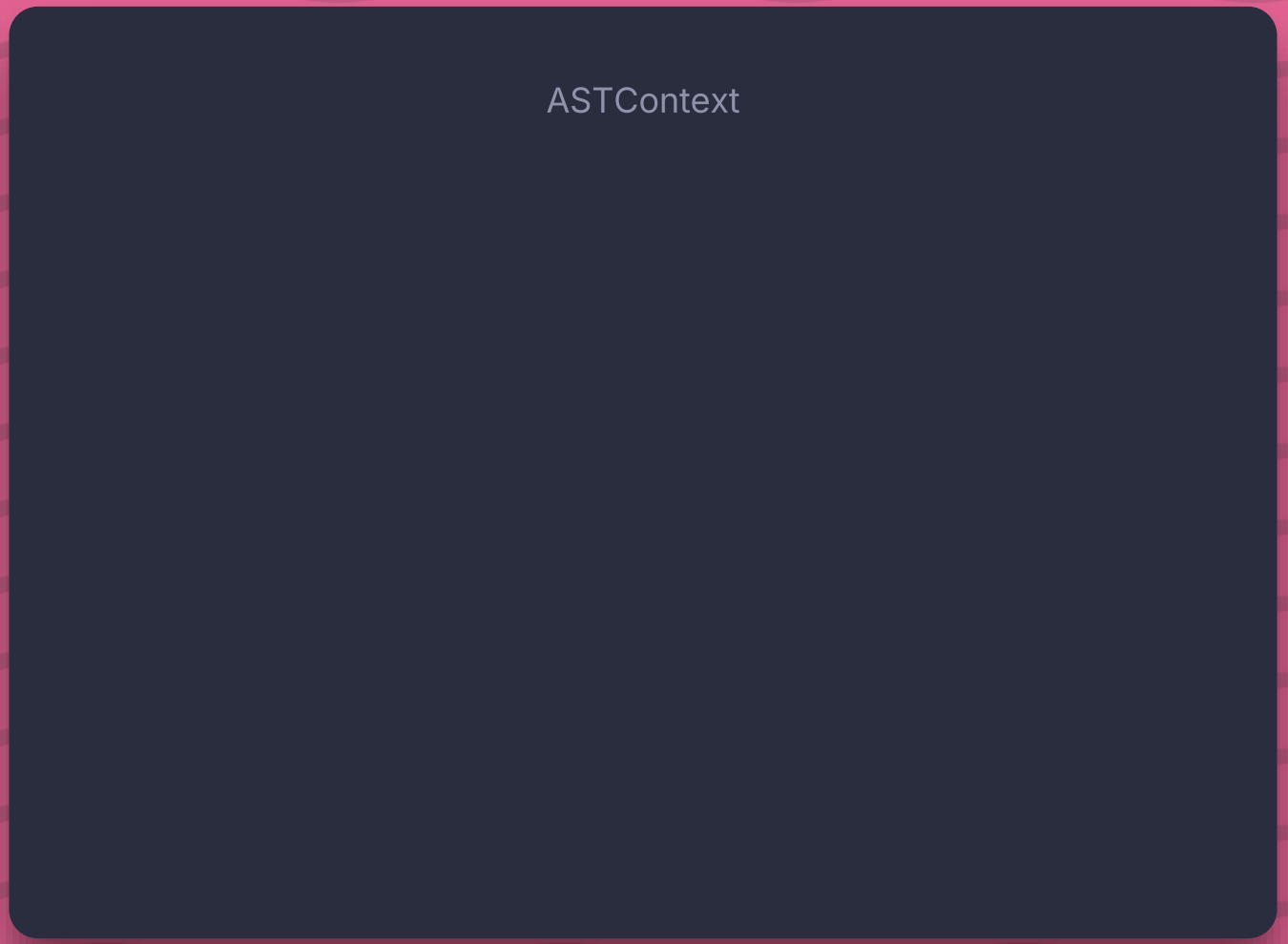
```
class Class {
  void privateMethod(int);

protected:
  void protectedMethod();

public:
  void publicMethod(void) {
    privateMethod(0);
  }
};
```

old/header.h

```
class Class {
  void privateMethod();

public:
  void publicMethod() {
    privateMethod();
  }
};
```

new/header.h

```
class Class {
  void privateMethod(int);

protected:
  void protectedMethod();

public:
  void publicMethod(void) {
    privateMethod(0);
  }
};
```


ASTContext

ASTContext

```
TranslationUnitDecl 0xA0
`-FunctionDecl 0xA1 main 'int ()'
  `-CompoundStmt 0xA2
    |-CallExpr 0xA3
    | `-ImplicitCastExpr 0xA4
    |   `-DeclRefExpr 0xA5 'eurollvm'
    `-ReturnStmt 0xA6
      `-IntegerLiteral 0xA7 'int' 0
```

ASTContext

```
TranslationUnitDecl 0xA0
`-FunctionDecl 0xA1 main 'int ()'
```

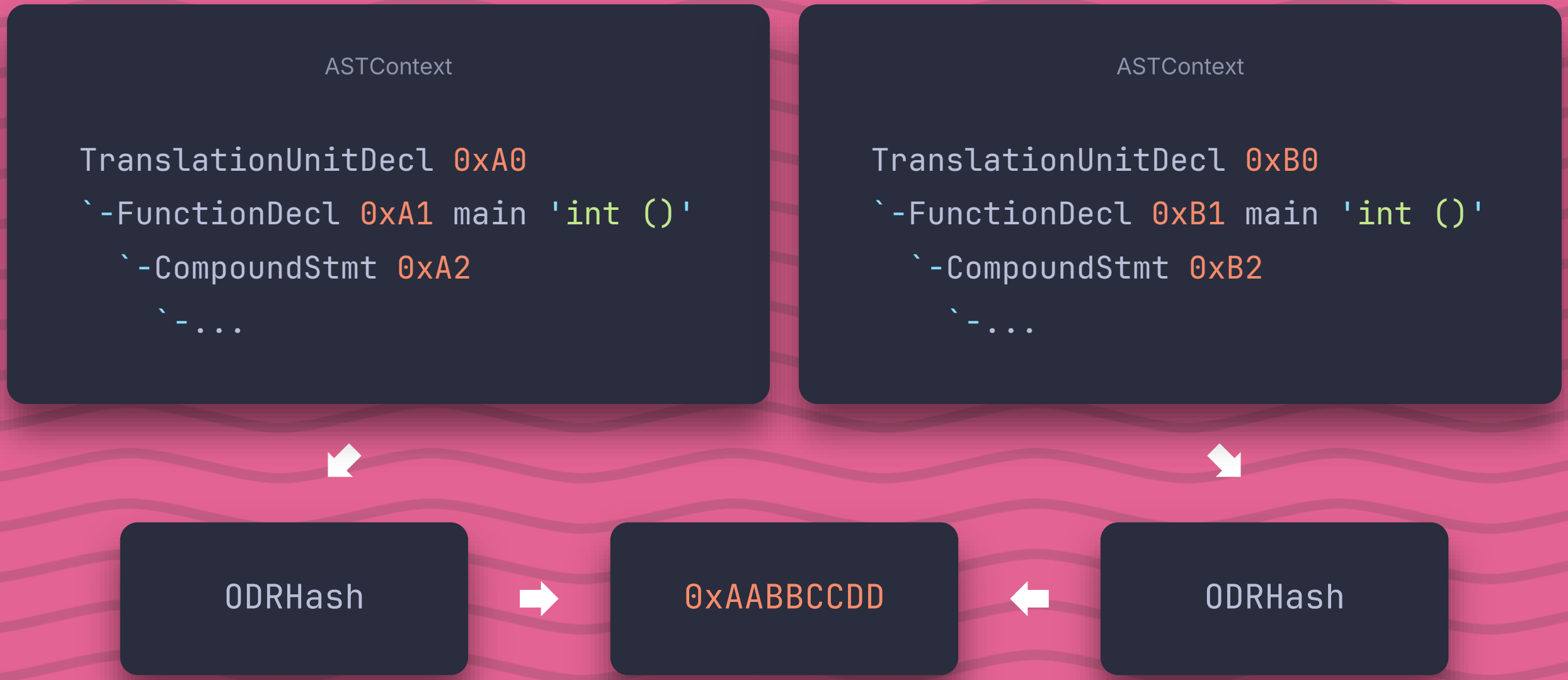
```
bool IsSameDecl (Decl *A,
                 Decl *B) {
  return A == B;
}
```

ASTContext

```
TranslationUnitDecl 0xA0
`-FunctionDecl 0xA1 main 'int ()'
  `-CompoundStmt 0xA2
    `-...
```

ASTContext

```
TranslationUnitDecl 0xB0
`-FunctionDecl 0xB1 main 'int ()'
  `-CompoundStmt 0xB2
    `-...
```


ASTContext
TranslationUnitDecl 0xA0
`-FunctionDecl 0xA1 main 'int ()'
`-CompoundStmt 0xA2
`-...
ASTContext
TranslationUnitDecl 0xB0
`-FunctionDecl 0xB1 main 'int ()'
`-CompoundStmt 0xB2
`-...
ODRHash
0xAABBCCDD
ODRHash

github.com/isuckatcs/abicorn-on-graduation-ceremony

Public

Star

It's exactly like those graduation gift balloons, except this is a Clang Tool that detects library-level API and ABI compatibility breaking changes based on source code. 🦄🎓🎈

C++ ☆ 2 Updated yesterday

old/mytalk.h

```
void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double);
```

new/mytalk.h

```
void abiBreaking(_Complex unsigned char, signed char,
                 __int128, __float128, long double, bool = false);
```

```
domjandaniel@eurollvm-2025:~$ abicorn
```

```
domjandaniel@eurollvm-2025:~$ abicorn --old old/mytalk.h
```

```
domjandaniel@eurollvm-2025:~$ abicorn --old old/mytalk.h --new new/mytalk.h
```

```
domjandaniel@eurollvm-2025:~$ abicorn --old old/mytalk.h --new new/mytalk.h
.../old/my_talk.cpp:5:6: warning: overload of 'abiBreaking' not found in the new
library [function-checker]
    5 | void abiBreaking(_Complex unsigned char, signed char, __int128, __float128,
      |      ^
.../new/my_talk.cpp:5:6: note: found similar function [function-checker]
    5 | void abiBreaking(_Complex unsigned char, signed char,
      |      ^
domjandaniel@eurollvm-2025:~$
```

Review #1B

I see that we in LLVM do ... libclang-abi-tests ...

Review #1A

One way to evaluate the tool would be to compare LLVM "dot" releases, which are supposed to be ABI compatible ...

Review #1C

Since the tool is based on AST comparison, it'd be also useful to know how the tool differs from the existing clang-diff tool that's already a part of LLVM.

```
domjandaniel@eurollvm-2025:~$ abicorn
```

```
domjandaniel@eurollvm-2025:~$ abicorn \
> --old $(find llvm-15-0-0/llvm/include/llvm/DebugInfo/Symbolize -type f)
```

```
domjandaniel@eurollvm-2025:~$ abicorn \
> --old $(find llvm-15-0-0/llvm/include/llvm/DebugInfo/Symbolize -type f) \
> --p-old=llvm-15-0-0/build
```

```
domjandaniel@eurollvm-2025:~$ abicorn \
> --old $(find llvm-15-0-0/llvm/include/llvm/DebugInfo/Symbolize -type f) \
> --p-old=llvm-15-0-0/build \
> --new $(find llvm-15-0-1/llvm/include/llvm/DebugInfo/Symbolize -type f) \
> --p-new=llvm-15-0-1/build
```

```
domjandaniel@eurollvm-2025:~$ abicorn \
> --old $(find llvm-15-0-0/llvm/include/llvm/DebugInfo/Symbolize -type f) \
> --p-old=llvm-15-0-0/build \
> --new $(find llvm-15-0-1/llvm/include/llvm/DebugInfo/Symbolize -type f) \
> --p-new=llvm-15-0-1/build
...
llvm-15-0-1/llvm/include/llvm/DebugInfo/Symbolize/MarkupFilter.h:140:19: warning:
field 'Symbolizer' is missing from the old library [field-checker]
  140 |   LLVMSymbolizer &Symbolizer;
      |                   ^
llvm-15-0-0/llvm/include/llvm/DebugInfo/Symbolize/MarkupFilter.h:33:3: warning:
overload of 'MarkupFilter' not found in the new library [method-checker]
   33 |   MarkupFilter(raw_ostream &OS, Optional<bool> ColorsEnabled = llvm::None);
      |   ^
llvm-15-0-1/llvm/include/llvm/DebugInfo/Symbolize/MarkupFilter.h:35:3: note:
found similar method [method-checker]
   35 |   MarkupFilter(raw_ostream &OS, LLVMSymbolizer &Symbolizer,
      |   ^
domjandaniel@eurollvm-2025:~$
```

llvm-15-0-0/.../MarkupFilter.h

```
class MarkupFilter {
public:
  MarkupFilter(raw_ostream &,
               Optional<bool> = None);

private:
  raw_ostream &OS;
  const bool ColorsEnabled;
  ...
};
```

llvm-15-0-1/.../MarkupFilter.h

```
class MarkupFilter {
public:
  MarkupFilter(raw_ostream &,
               LLVMSymbolizer &,
               Optional<bool> = None);

private:
  raw_ostream &OS;
  LLVMSymbolizer &Symbolizer;
  const bool ColorsEnabled;
  ...
};
```

TooSimpleToGoWrong.cpp

```cpp
#include <llvm/DebugInfo/Symbolize/MarkupFilter.h>

int main() {
  llvm::symbolize::MarkupFilter MF(llvm::errs());

  return 0;
}
```

```
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.0
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.0
domjandaniel@eurollvm-2025:~$ clang++ TooSimpleToGoWrong.cpp \
> $(llvm-config --cxxflags --ldflags) -l:libLLVM-15.so -o this-is-fine
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.0
domjandaniel@eurollvm-2025:~$ clang++ TooSimpleToGoWrong.cpp \
> $(llvm-config --cxxflags --ldflags) -l:libLLVM-15.so -o this-is-fine
domjandaniel@eurollvm-2025:~$ ./this-is-fine
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.0
domjandaniel@eurollvm-2025:~$ clang++ TooSimpleToGoWrong.cpp \
> $(llvm-config --cxxflags --ldflags) -l:libLLVM-15.so -o this-is-fine
domjandaniel@eurollvm-2025:~$ ./this-is-fine
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.1
domjandaniel@eurollvm-2025:~$
```

```
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.0
domjandaniel@eurollvm-2025:~$ clang++ TooSimpleToGoWrong.cpp \
> $(llvm-config --cxxflags --ldflags) -l:libLLVM-15.so -o this-is-fine
domjandaniel@eurollvm-2025:~$ ./this-is-fine
domjandaniel@eurollvm-2025:~$ please-install-me-llvm-util --version 15.0.1
domjandaniel@eurollvm-2025:~$ ./this-is-fine 🔥🐶💬☕🔥
./this-is-fine: symbol lookup error: ./this-is-fine: undefined symbol: _ZN4llvm9symb
olize12MarkupFilterC1ERNS_11raw_ostreamENS_8OptionalIbEE, version LLVM_15
domjandaniel@eurollvm-2025:~$
```

Q/A