2023

Interactive Tooling Support for the Migration to Strong Types

Richárd Szalay

C++ now

Interactive Tooling Support for the Migration to Strong Types

Richárd Szalay

Eötvös Loránd University
Faculty of Informatics

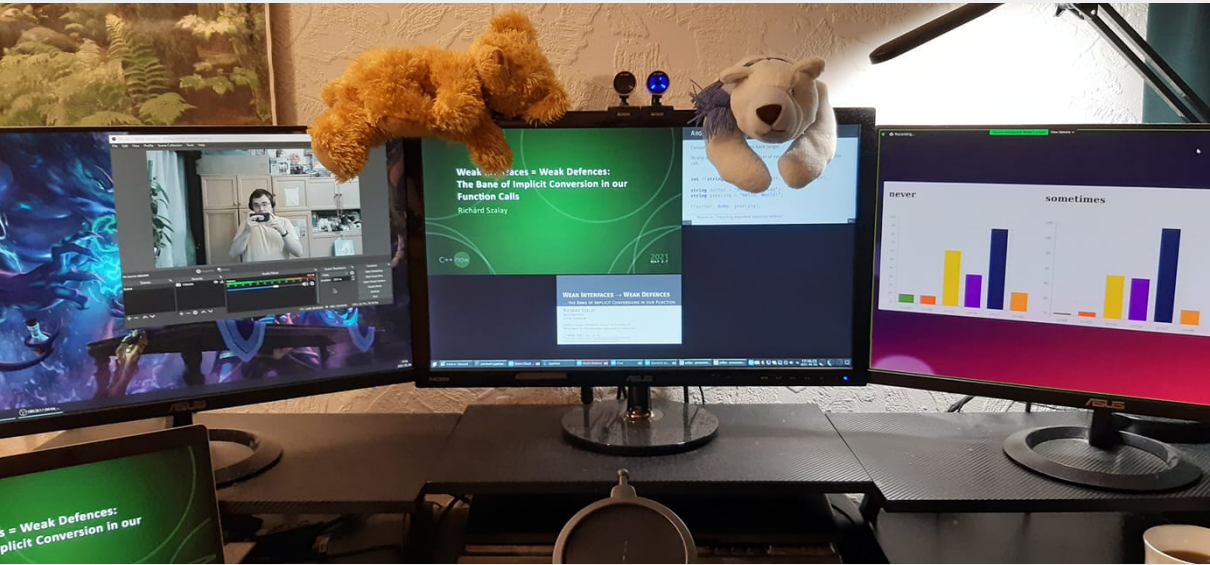
Department of Programming Languages & Compilers,
ELTE – Eötvös Loránd University,
Budapest, Hungary

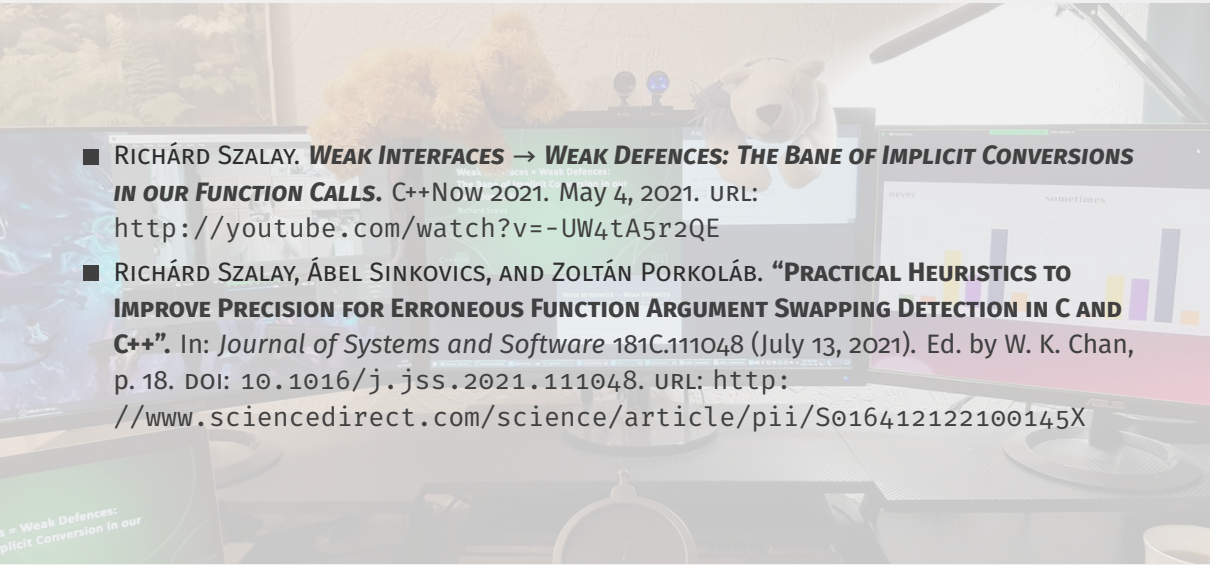
2023. 05. 10.

Prepared with the professional support of the Doctoral Student Scholarship Program of the Co-operative Doctoral Program of the Ministry of Innovation and Technology financed from the National Research, Development and Innovation Fund.

1. Introduction
   - Strong typing
   - Type migration – by hand
2. Overview
   - Initial taint
   - Propagation
   - Rewriting the code
3. In detail
   - What is a “fictive type”?
   - Code generation
   - ~~Problems~~ Design decisions
4. Tooling & “Demo”
   - Initial setup
   - Propagation
   - “Old” infrastructure
   - Inlining
   - Iterative driver


SNOWMASS VILLAGE
POPULATION 2863
ELEVATION 8410
INCORPORATED 1977
DOGS REGISTERED 210
TOTAL 13460


Weak ... aces = Weak Defences:
The Bane of Implicit Conversion in our Function Calls
Richárd Szalay
C++ Now
2021
Weak Interfaces → Weak Defences
never
sometimes
s = Weak Defences:
plicit Conversion in our



- Richárd Szalay. ***Weak Interfaces → Weak Defences: The Bane of Implicit Conversions in our Function Calls.*** C++Now 2021. May 4, 2021. URL: http://youtube.com/watch?v=-UW4tA5r2QE
- Richárd Szalay, Ábel Sinkovics, and Zoltán Porkoláb. **"Practical Heuristics to Improve Precision for Erroneous Function Argument Swapping Detection in C and C++".** In: *Journal of Systems and Software* 181C.111048 (July 13, 2021). Ed. by W. K. Chan, p. 18. DOI: 10.1016/j.jss.2021.111048. URL: http://www.sciencedirect.com/science/article/pii/S016412122100145X

src/classify/adaptmatch.cpp

```
1738  * case of error.
1739  */
1740 int Classify::MakeNewTemporaryConfig(ADAPT_TEMPLATES Templates,
1741                                      CLASS_ID ClassId,
1742                                      int FontinfoId,
1743                                      int NumFeatures,
1744                                      INT_FEATURE_ARRAY Features,
1745                                      FEATURE_SET FloatFeatures) {
```

1 the first parameter in this range is 'ClassId'

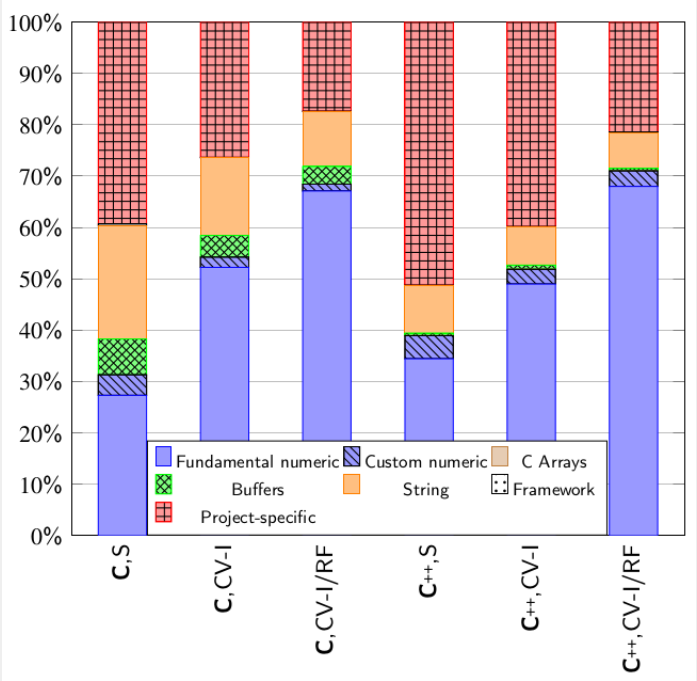
3 after resolving type aliases, type of parameter 'ClassId' is 'int'

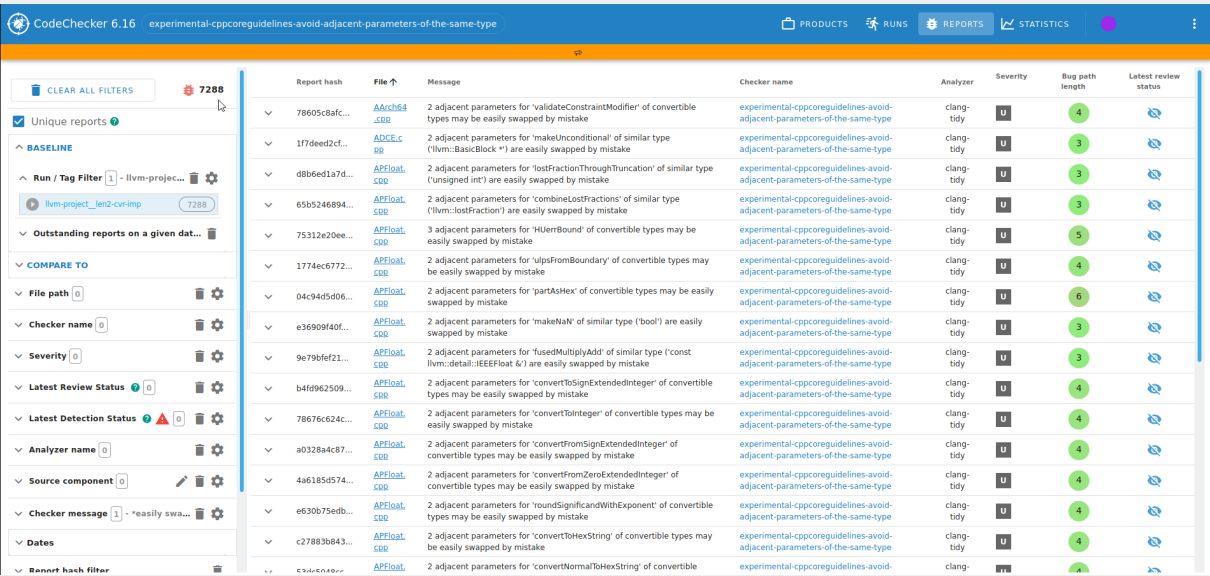
6 3 adjacent parameters for 'MakeNewTemporaryConfig' of similar type are easily swapped by mistake

4 after resolving type aliases, type of parameter 'FontinfoId' is 'int'

2 the last parameter in this range is 'NumFeatures'

5 after resolving type aliases, type of parameter 'NumFeatures' is 'int'


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Fundamental numeric
Custom numeric
C Arrays
Buffers
String
Framework
Project-specific
C,S
C,CV-I
C,CV-I/RF
C++,S
C++,CV-I
C++,CV-I/RF


CodeChecker 6.16
experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type
PRODUCTS
RUNS
REPORTS
STATISTICS
CLEAR ALL FILTERS
7288
Unique reports
BASELINE
Run / Tag Filter 1 - llvm-projec...
llvm-project__len2-cvr-imp
7288
Outstanding reports on a given dat...
COMPARE TO
File path 0
Checker name 0
Severity 0
Latest Review Status 0
Latest Detection Status 0
Analyzer name 0
Source component 0
Checker message 1 - *easily swa...
Dates
Report hash filter
Report hash
File
Message
Checker name
Analyzer
Severity
Bug path length
Latest review status
78605c8afc... AArch64.cpp 2 adjacent parameters for 'validateConstraintModifier' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
1f7deed2cf... ADCE.cpp 2 adjacent parameters for 'makeUnconditional' of similar type ('llvm::BasicBlock *') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
d8b6ed1a7d... APFloat.cpp 2 adjacent parameters for 'lostFractionThroughTruncation' of similar type ('unsigned int') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
65b5246894... APFloat.cpp 2 adjacent parameters for 'combineLostFractions' of similar type ('llvm::lostFraction') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
75312e20ee... APFloat.cpp 3 adjacent parameters for 'HUerrBound' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 5
1774ec6772... APFloat.cpp 2 adjacent parameters for 'ulpsFromBoundary' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
04c94d5d06... APFloat.cpp 2 adjacent parameters for 'partAsHex' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 6
e36909f40f... APFloat.cpp 2 adjacent parameters for 'makeNaN' of similar type ('bool') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
9e79bfef21... APFloat.cpp 2 adjacent parameters for 'fusedMultiplyAdd' of similar type ('const llvm::detail::IEEEFloat &') are easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 3
b4fd962509... APFloat.cpp 2 adjacent parameters for 'convertToSignExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
78676c624c... APFloat.cpp 2 adjacent parameters for 'convertToInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
a0328a4c87... APFloat.cpp 2 adjacent parameters for 'convertFromSignExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
4a6185d574... APFloat.cpp 2 adjacent parameters for 'convertFromZeroExtendedInteger' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
e630b75edb... APFloat.cpp 2 adjacent parameters for 'roundSignificandWithExponent' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
c27883b843... APFloat.cpp 2 adjacent parameters for 'convertToHexString' of convertible types may be easily swapped by mistake experimental-cppcoreguidelines-avoid-adjacent-parameters-of-the-same-type clang-tidy U 4
APFloat. 2 adjacent parameters for 'convertNormalToHexString' of convertible experimental-cppcoreguidelines-avoid- clang-

Reactions

CppCast ep. 322…
Jason Turner's eBook *C++ Best Practices*

"That sounds like a potentially extremely helpful static analysis."

Reactions

disable bugprone-easily-swappable-parameters

Browse files

```
clang-tidy can be a nugget sometimes x3
```

main

raisinware committed on Apr 4 — 1 parent f286693 commit 406ef68

Showing **1 changed file** with **1 addition** and **1 deletion**.

Split | Unified

2 .clang-tidy

```
...  @@ -1,5 +1,5 @@
1    ---                                                                  1    ---
2  - Checks: '-*,bugprone-*,clang-analyzer-*,-clang-analyzer-cplusplus*,clang-  2  + Checks: '-*,bugprone-*,clang-analyzer-*,-clang-analyzer-cplusplus*,clang-
     diagnostic-*,modernize-*,performance-*,portability-*,readability-*'            diagnostic-*,modernize-*,performance-*,portability-*,readability-*,-bugprone-easily-
                                                                                    swappable-parameters'
3    FormatStyle: none                                                    3    FormatStyle: none
4    UseColor: true                                                       4    UseColor: true
5    ...                                                                  5    ...
```

Reactions

Sentiment	#
	3
	64 (26 comments)
	5
	11
	9 (7 comments)

This talk

Tool demo
Arguments
Arguments

Questions
Philosophy
Arguments & Parameters

This talk

Tool demo
Arguments
Arguments

Questions
Philosophy
Arguments & Parameters & **Everything else?**

1 Introduction

- Strong typing
  - What? & How?
  - Negative overhead?
  - *Some* existing libraries
- Type migration – by hand

2 Overview

3 In detail

4 Tooling & “Demo”

Stronger(?) types

```
int threshold(int Temperature) { return 3 * Temperature; }

int SensorTemp = /* … */;
int T2 = threshold(SensorTemp);
print(T2);
```

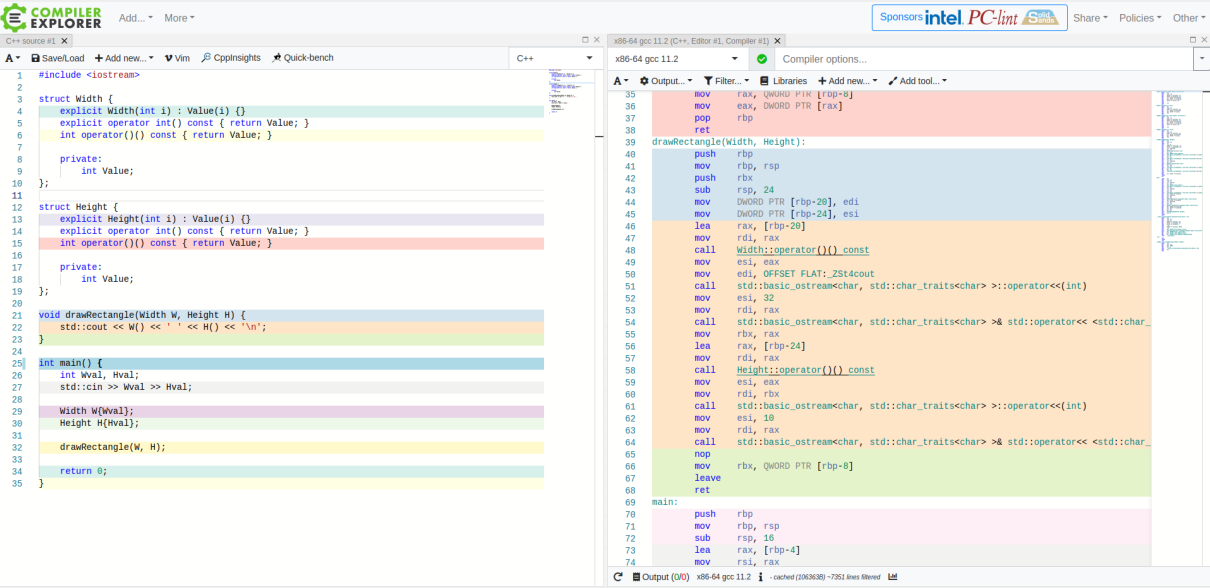
```
class temperature { /* ??? */ };
/* ??? */
temperature threshold(temperature T) { return 3 * T; }

temperature Sensor = /* … */;
temperature T2 = threshold(Sensor);
print(static_cast<int>(T2));
```

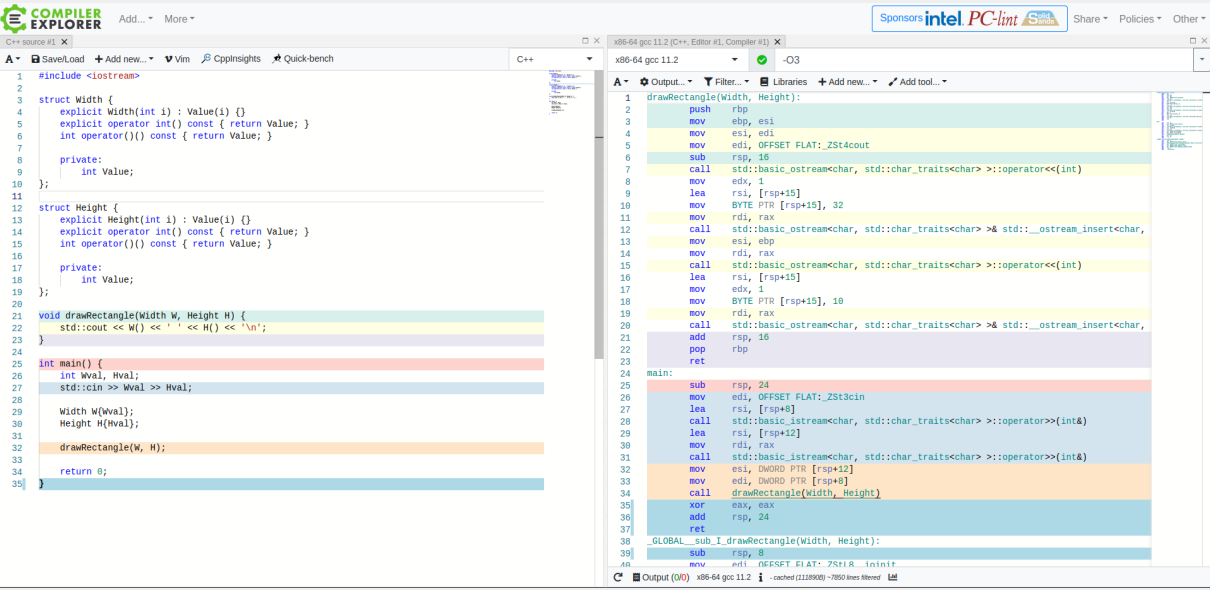
Strong typedef I

```
void drawBad(int width, int height);

struct Width {
  int value;

  explicit Width(int v) : value(v) {}
  explicit operator int() const {
    return value;
  }
  int operator()() const {
    return value;
  }
};
struct Height { /* Analogous… */ };
```

Strong typedef II

```
void draw(Width w, Height h) {
  int wi1 = w.value, he1 = h.value;
  int wi2 = w(), he2 = h();
  int wi3 = (int)w, he3 = static_cast<int>(h);
  int wi4{w}, he4{h};
}
// ↯ error: no implicit conversion from 'int' to 'Width'
draw(1, 2, RED);
// ↯ compile error, type mismatch
draw(Height{2}, Width{1}, RED);

draw(Width{1}, Height{2}, RED); // ✓ Works!
```


COMPILER EXPLORER
Add...
More
Sponsors intel PC-lint
Share
Policies
Other
C++ source #1
Save/Load
Add new...
Vim
CppInsights
Quick-bench
C++
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1)
x86-64 gcc 11.2
Compiler options...
Output...
Filter...
Libraries
Add new...
Add tool...
#include <iostream>
struct Width {
explicit Width(int i) : Value(i) {}
explicit operator int() const { return Value; }
int operator()() const { return Value; }
private:
int Value;
};
struct Height {
explicit Height(int i) : Value(i) {}
explicit operator int() const { return Value; }
int operator()() const { return Value; }
private:
int Value;
};
void drawRectangle(Width W, Height H) {
std::cout << W() << ' ' << H() << '\n';
}
int main() {
int Wval, Hval;
std::cin >> Wval >> Hval;
Width W{Wval};
Height H{Hval};
drawRectangle(W, H);
return 0;
}
mov rax, QWORD PTR [rbp-8]
mov eax, DWORD PTR [rax]
pop rbp
ret
drawRectangle(Width, Height):
push rbp
mov rbp, rsp
push rbx
sub rsp, 24
mov DWORD PTR [rbp-20], edi
mov DWORD PTR [rbp-24], esi
lea rax, [rbp-20]
mov rdi, rax
call Width::operator()() const
mov esi, eax
mov edi, OFFSET FLAT:_ZSt4cout
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov esi, 32
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_
mov rbx, rax
lea rax, [rbp-24]
mov rdi, rax
call Height::operator()() const
mov esi, eax
mov rdi, rbx
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov esi, 10
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_
nop
mov rbx, QWORD PTR [rbp-8]
leave
ret
main:
push rbp
mov rbp, rsp
sub rsp, 16
lea rax, [rbp-4]
mov rsi, rax
Output (0/0)
x86-64 gcc 11.2
- cached (1063638) - 7351 lines filtered


COMPILER EXPLORER
Add...
More
Sponsors intel PC-lint
Share
Policies
Other
C++ source #1
Save/Load
Add new...
Vim
CppInsights
Quick-bench
C++
x86-64 gcc 11.2 (C++, Editor #1, Compiler #1)
x86-64 gcc 11.2
-O3
Output...
Filter...
Libraries
Add new...
Add tool...
Output (0/0)
x86-64 gcc 11.2

Strong interface I

```
struct Width {
  int value;
  explicit Width(int v) : value(v) {}
  explicit operator int() const {
    return value;
  }
  int operator()() const {
    return value;
  }
};

struct Height { /* Analogous… */ };
struct Area   { /* Analogous… */ };
```

Strong interface II

```
Area operator*(Width w, Height h) {
  return Area{w.get() * h.get()};
}
```

Strong *type* I

```cpp
#include <chrono>
using namespace std::chrono;
using namespace std::literals;

// Bad: prone to bad order of arguments.
bool submit_at_1(int year, int month, int day,
                 int hour, int minute, int second);

// Bad: "seconds" is not descriptive.
bool submit_at_2(double seconds);

// Order of arguments mixed up.
submit_at_1(23, 59, 59, 2023, 1, 31);
```

Strong *type* II

```
// Semantically incorrect, yet compiles.
submit_at_2(get_milliseconds());

bool submit_at_good(time_point<system_clock, seconds> T) {
  auto DLDay = 2023y / January / 31;
  auto DLSecond = 24h - 1s; // = 86 399 sec
  auto AOEDeadline = zoned_time(
    "Etc/GMT+12", DLDay + DLSecond);
  return T <= AOEDeadline.get_sys_time();
}

submit_at_good(2022); // ↯ compile error: no conversion.
submit_at_good(system_clock::now());
```

C++ source #1

```
C++
1  int f(int& x, int& y) {
2      x = 512;
3      y = 42;
4      return x / y;
5  }
6
7  int main()
8  {
9      int x = 0;
10     return f(x, x);
11 }
```

(6, 1)

C++ source #2

```
C++
1  struct IntWrap
2  {
3      int V;
4      IntWrap(int v) : V(v) {}
5      IntWrap& operator=(int v) { V = v; return *this; }
6      operator int() const { return V; }
7  };
8  struct Nom : IntWrap {};
9  struct Den : IntWrap {};
10
11 int f(Nom& n, Den& d)
12 {
13     n.V = 512;
14     d.V = 42;
15     return n.V / d.V;
16 }
17
18 int main()
19 {
20     Nom N{0}; Den D{0};
21     return f(N, D);
22 }
```

x86-64 gcc 12.2 (Editor #1)

x86-64 gcc 12.2 -O0

Output... Filter... Libraries + Add new... Add tool...

```
1  f(int&, int&):
2          push    rbp
3          mov     rbp, rsp
4          mov     QWORD PTR [rbp-8], rdi
5          mov     QWORD PTR [rbp-16], rsi
6          mov     rax, QWORD PTR [rbp-8]
7          mov     DWORD PTR [rax], 512
8          mov     rax, QWORD PTR [rbp-16]
9          mov     DWORD PTR [rax], 42
10         mov     rax, QWORD PTR [rbp-8]
11         mov     eax, DWORD PTR [rax]
12         mov     rdx, QWORD PTR [rbp-16]
13         mov     ecx, DWORD PTR [rdx]
14         cdq
15         idiv    ecx
16         pop     rbp
17         ret
18 main:
19         push    rbp
20         mov     rbp, rsp
21         sub     rsp, 16
22         mov     DWORD PTR [rbp-4], 0
23         lea     rdx, [rbp-4]
24         lea     rax, [rbp-4]
25         mov     rsi, rdx
26         mov     rdi, rax
27         call    f(int&, int&)
28         nop
29         leave
30         ret
```

Output (0/0) x86-64 gcc 12.2 - cached (4590B) ~273 lines filtered Compiler License

Output of x86-64 gcc 12.2 (Compiler #1)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 1
```

x86-64 gcc 12.2 (Editor #2)

x86-64 gcc 12.2 -O0

Output... Filter... Libraries + Add new... Add tool...

```
1  IntWrap::IntWrap(int) [base object constructor]:
2          push    rbp
3          mov     rbp, rsp
4          mov     QWORD PTR [rbp-8], rdi
5          mov     DWORD PTR [rbp-12], esi
6          mov     rax, QWORD PTR [rbp-8]
7          mov     edx, DWORD PTR [rbp-12]
8          mov     DWORD PTR [rax], edx
9          nop
10         pop     rbp
11         ret
12 f(Nom&, Den&):
13         push    rbp
14         mov     rbp, rsp
15         mov     QWORD PTR [rbp-8], rdi
16         mov     QWORD PTR [rbp-16], rsi
17         mov     rax, QWORD PTR [rbp-8]
18         mov     DWORD PTR [rax], 512
19         mov     rax, QWORD PTR [rbp-16]
20         mov     DWORD PTR [rax], 42
21         mov     rax, QWORD PTR [rbp-8]
22         mov     eax, DWORD PTR [rax]
23         mov     rdx, QWORD PTR [rbp-16]
24         mov     ecx, DWORD PTR [rdx]
25         cdq
26         idiv    ecx
27         pop     rbp
28         ret
29 main:
30         push    rbp
31         mov     rbp, rsp
32         sub     rsp, 16
33         lea     rax, [rbp-4]
34         mov     esi, 0
35         mov     rdi, rax
36         call    IntWrap::IntWrap(int) [base object const
37         lea     rax, [rbp-8]
38         mov     esi, 0
39         mov     rdi, rax
40         call    IntWrap::IntWrap(int) [base object const
41         lea     rdx, [rbp-8]
```

Output (0/0) x86-64 gcc 12.2 - cached (10646B) ~685 lines filtered Compiler License

Output of x86-64 gcc 12.2 (Compiler #2)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 12
```

C++ source #1

```
int f(int& x, int& y) {
    x = 512;
    y = 42;
    return x / y;
}

int main()
{
    int x = 0;
    return f(x, x);
}
```

C++ source #2

```
struct IntWrap
{
    int V;
    IntWrap(int v) : V(v) {}
    IntWrap& operator=(int v) { V = v; return *this; }
    operator int() const { return V; }
};
struct Nom : IntWrap {};
struct Den : IntWrap {};

int f(Nom& n, Den& d)
{
    n.V = 512;
    d.V = 42;
    return n.V / d.V;
}

int main()
{
    Nom N{0}; Den D{0};
    return f(N, D);
}
```

x86-64 gcc 12.2 (Editor #1)

x86-64 gcc 12.2 -O3

Output... Filter... Libraries Add new... Add tool...

```
f(int&, int&):
        mov     DWORD PTR [rdi], 512
        mov     DWORD PTR [rsi], 42
        movsx   rax, DWORD PTR [rdi]
        mov     rdx, rax
        imul    rax, rax, 818089009
        sar     edx, 31
        sar     rax, 35
        sub     eax, edx
        ret
main:
        mov     eax, 1
        ret
```

Output (0/0) x86-64 gcc 12.2 - cached (53358) ~337 lines filtered Compiler License

Output of x86-64 gcc 12.2 (Compiler #1)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 1
```

x86-64 gcc 12.2 (Editor #2)

x86-64 gcc 12.2 -O3

Output... Filter... Libraries Add new... Add tool...

```
f(Nom&, Den&):
        mov     DWORD PTR [rdi], 512
        mov     eax, 12
        mov     DWORD PTR [rsi], 42
        ret
main:
        mov     eax, 12
        ret
```

Output (0/0) x86-64 gcc 12.2 - cached (124648) ~865 lines filtered Compiler License

Output of x86-64 gcc 12.2 (Compiler #2)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 12
```

C++ source #1

```
int f(int& x, int& y) {
    x = 512;
    y = 42;
    return x / y;
}

int main()
{
    int x = 0;
    return f(x, x);
}
```

C++ source #2

```
struct IntWrap
{
    int V;
    IntWrap(int v) : V(v) {}
    IntWrap& operator=(int v) { V = v; return *this; }
    operator int() const { return V; }
};
struct Nom : IntWrap {};
struct Den : IntWrap {};

int f(Nom& n, Den& d)
{
    n.V = 512;
    d.V = 42;
    return n.V / d.V;
}

int main()
{
    Nom N{0}; Den D{0};
    return f(N, D);
}
```

x86-64 clang 16.0.0 (Editor #1)

x86-64 clang 16.0.0 -O3

Output... Filter... Libraries Add new... Add tool...

```
f(int&, int&):                          # @f(int&,
        mov     dword ptr [rdi], 512
        mov     dword ptr [rsi], 42
        movsxd  rax, dword ptr [rdi]
        imul    rax, rax, 818089009
        mov     rcx, rax
        shr     rcx, 63
        sar     rax, 35
        add     eax, ecx
        ret
main:                                   # @main
        mov     eax, 1
        ret
```

Output (0/0) x86-64 clang 16.0.0 - cached (12714B) ~247 lines filtered Compiler License

Output of x86-64 clang 16.0.0 (Compiler #1)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 1
```

x86-64 clang 16.0.0 (Editor #2)

x86-64 clang 16.0.0 -O3

Output... Filter... Libraries Add new... Add tool...

```
f(Nom&, Den&):                          # @f(Nom&, Den&)
        mov     dword ptr [rdi], 512
        mov     dword ptr [rsi], 42
        movsxd  rax, dword ptr [rdi]
        imul    rax, rax, 818089009
        mov     rcx, rax
        shr     rcx, 63
        sar     rax, 35
        add     eax, ecx
        ret
main:                                   # @main
        mov     eax, 12
        ret
```

Output (0/0) x86-64 clang 16.0.0 - cached (24774B) ~460 lines filtered Compiler License

Output of x86-64 clang 16.0.0 (Compiler #2)

Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 12
```

Chrono-like physics[1]

```
#include <units/isq/si/area.h>
#include <units/isq/si/frequency.h>
#include <units/isq/si/length.h>
#include <units/isq/si/speed.h>
#include <units/isq/si/time.h>

using namespace units::isq::si::references;

// simple numeric operations
static_assert(10 * km / 2 == 5 * km);

// unit conversions
static_assert(1 * h == 3600 * s);
static_assert(1 * km + 1 * m == 1001 * m);
```

```
// dimension conversions
inline constexpr auto kmph = km / h;
static_assert(1 * km / (1 * s) ==
              1000 * (m / s));
static_assert(2 * kmph * (2 * h) == 4 * km);
static_assert(2 * km / (2 * kmph) == 1 * h);

static_assert(2 * m * (3 * m) == 6 * m2);

static_assert(10 * km / (5 * km) == 2);

static_assert(1000 / (1 * s) == 1 * kHz);
```

[1] Mateusz Pusz. ***Implementing Physical Units Library for C++.*** C++Now 2019. May 9, 2019. URL: youtube.com/watch?v=wKchCktZPHU.

Simple generic wrappers[2]

```
using namespace pssst;

struct Voltage : Linear<double,Voltage,Out> {};
struct Current : Linear<double,Current,Out> {};
struct Resistance
    : Linear<double,Resistance,Out> {};

Voltage operator""_V(long double val) {
    return Voltage{static_cast<double>(val)};
}
Current operator/(Voltage v, Resistance r) {
    auto [vv] = v;    auto [rr] = r;
    return { vv / rr } ;
}
Resistance operator/(Voltage v, Current c) {
    auto [vv] = v;    auto [cc] = c;
    return { vv / cc } ;
}
```

```
Voltage operator*(Resistance r, Current c) {
    auto [rr] = r;    auto [cc] = c;
    return { rr * cc } ;
}
Voltage operator*(Current c, Resistance r) {
    auto [rr] = r;    auto [cc] = c;
    return { rr * cc } ;
}

auto result1 = 10_V/100_Ohm;
ASSERT_EQUAL(0.1_A, result1);

auto result2 = 10_V/0.1_A;
ASSERT_EQUAL(100_Ohm, result2);
```

[2] Peter Sommerlad. ***Simplest Strong Typing instead of Language Proposal (P0109).*** C++Now 2021. May 6, 2021. URL: youtube.com/watch?v=ABkxMSbejZI.

Established algebræ[3]

```
absl::Duration absl::Minutes(double);
absl::Duration absl::Seconds(double);
absl::Duration absl::Milliseconds(double);
```

(a) `absl::Duration` factory functions: convert numeric values to `absl::Duration` value at the given scale.

```
double absl::ToDoubleMinutes(absl::Duration);
double absl::ToDoubleSeconds(absl::Duration);
double absl::ToDoubleMilliseconds(absl::Duration);
```

(b) `absl::Duration` conversion functions: convert an `absl::Duration` to the indicated scale.

```
absl::Time absl::FromUnixMinutes(int64_t);
absl::Time absl::FromUnixSeconds(int64_t);
absl::Time absl::FromUnixMillis(int64_t);
```

(c) `absl::Time` factory functions: convert a numeric value at the given scale since the Unix epoch to an `absl::Time` representing that time instant.

```
int64_t absl::FromUnixMinutes(absl::Time);
int64_t absl::FromUnixSeconds(absl::Time);
int64_t absl::FromUnixMillis(absl::Time);
```

(d) `absl::Time` conversion functions: take an `absl::Time` and return a numeric value at the given scale since the Unix epoch.

TABLE I: Selections from the Abseil Time API

Expression	Result
Time + Duration	Time
Time − Duration	Time
Time + Time	*Undefined*
Time − Time	Duration
Duration + Duration	Duration
Duration − Duration	Duration
Duration + Time	Time
Duration − Time	*Undefined*

TABLE II: Addition and Subtraction operations for Abseil Time types

```
void func(absl::Duration d) {
  int x = get_deadline();
  if (d < absl::Seconds(x)) ...

  x += 60;
  x *= 2;
  int y = x;
}
```

(a) Before transformation

```
void func(absl::Duration d) {
  absl::Duration x = absl::Seconds(get_deadline());
  if (d < x) ...

  x += absl::Seconds(60);
  x *= 2;
  int y = absl::ToInt64Seconds(x);
}
```

(b) After *DurationLocalVariable* has been applied

Listing 5: *DurationLocalVariable* transformation example

[3] Hyrum K. Wright. **"Incremental Type Migration Using Type Algebra"**. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Sept. 2020, pp. 756–765. DOI: 10.1109/ICSME46990.2020.00085. URL: ieeexplore.ieee.org/document/9240711.

Assume that you have already...

- Some existing software project
- A well-designed Strong Type library

Assume that you have already...

- Some existing software project
- A well-designed Strong Type library

Let's refactor!

Let's refactor...

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

int ST = readSensor(0);
int T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

“Temperature” STRONG TYPEDEF

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

int ST = readSensor(0);
int T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;
};
```

“Temperature” STRONG TYPEDEF

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

int ST = readSensor(0);
int T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  Temperature(int V) : Value(V) {}
  operator int() const { return Value; }
};
```

Inject new type

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

Temperature ST = readSensor(0);
int T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  Temperature(int V) : Value(V) {}
  operator int() const { return Value; }
};
```

Where to go?

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

Temperature ST = readSensor(0);
int T2 = threshold(ST); // ⟵ Implicit conversion.

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  Temperature(int V) : Value(V) {}
  operator int() const { return Value; }
};
```

explicit

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

Temperature ST = readSensor(0);
int T2 = threshold(ST); // ⟵ Implicit conversion?

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};
```

Errors!

```
int readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

int threshold(int T) { return 3 * T; }

// ----- 8< ----- >8 -----

Temperature ST = readSensor(0);
int T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};
```

```
error: candidate function not viable:
  no known conversion
  from 'Temperature' to 'int'
```

Change API...

```
Temperature readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

Temperature threshold(Temperature T) {
  return 3 * T;
}
// ----- 8< ----- >8 -----
Temperature ST = readSensor(0);
Temperature T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};
```

```
error: invalid operands to
  binary expression
  ('int' and 'Temperature')
```

New operations

```
Temperature readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

Temperature threshold(Temperature T) {
  return 3 * T;
}
// ----- 8< ----- >8 -----
Temperature ST = readSensor(0);
Temperature T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};

Temperature operator*(int A,
                      Temperature T) {
  return Temperature{
    A * static_cast<int>(T)};
}
```

Is this *really* "strong typing"?

```
Temperature readSensor(int ID);
void print(...);
#define ICON_COOLING (1 << 8)
#define ICON_HEATING (1 << 16)

Temperature threshold(Temperature T) {
  return 3 * T;
}
// ----- 8< ----- >8 -----
Temperature ST = readSensor(0);
Temperature T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? ICON_HEATING : ICON_COOLING);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};

Temperature operator*( int A ,
                       Temperature T) {
  return Temperature{
    A * static_cast<int>(T)};
}
```

Anyway...

```
Temperature readSensor(int ID);
void print(...);
enum Icon {
  Cooling = 1 <<  8,
  Heating = 1 << 16,
};

Temperature threshold(Temperature T) {
  return 3 * T;
}
// ----- 8< ----- >8 -----
Temperature ST = readSensor(0);
Temperature T2 = threshold(ST);

print(ST, T2,
  ST < T2 ? Cooling : Heating );
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V)
    : Value(V) {}
  explicit operator int() const {
    return Value;
  }
};

Temperature operator*(int A,
                      Temperature T) {
  return Temperature{
    A * static_cast<int>(T)};
}
```

Semi-proper I/O

```
#include <iostream>
Temperature readSensor(int ID);
enum Icon {
  Cooling = 1 <<  8,
  Heating = 1 << 16,
};

Temperature threshold(Temperature T) {
  return 3 * T;
}
// ----- 8< ----- >8 -----
Temperature ST = readSensor(0);
Temperature T2 = threshold(ST);

screen << ST << T2 <<
  (ST < T2 ? Cooling : Heating);
```

```
struct Temperature {
  int Value;

  Temperature() : Value({}) {}
  explicit Temperature(int V);
  explicit operator int() const;
};
Temperature operator*(int A,
                      Temperature T);

error: invalid operands to
  binary expression
  ('ostream' and 'Temperature')
note: (×10^100) candidate function not viable...
```

```
/usr/include/c++/11/ostream:245:7: note: candidate: ‘std::basic_ostream<_CharT, _Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(constvoid*) [with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT, _Traits>::__ostream_type = std::basic_ostream<char>]’
  245 |       operator<<(const void* __p)
      |       ^~~~~~~~
/usr/include/c++/11/ostream:245:30: note:   no known conversion for argument 1 from ‘Temperature’ to ‘const void*’
  245 |       operator<<(const void* __p)
      |                  ~~~~~~~~~~~~^~~
/usr/include/c++/11/ostream:250:7: note: candidate: ‘std::basic_ostream<_CharT, _Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(std::nullptr_t) [with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT, _Traits>::__ostream_type = std::basic_ostream<char>; std::nullptr_t= std::nullptr_t]’
  250 |       operator<<(nullptr_t)
      |       ^~~~~~~~
/usr/include/c++/11/ostream:250:18: note:   no known conversion for argument 1 from ‘Temperature’ to ‘std::nullptr_t’
  250 |       operator<<(nullptr_t)
      |                  ^~~~~~~~~
In file included from /usr/include/c++/11/ostream:829,
                 from /usr/include/c++/11/iostream:39,
                 from a.cpp:1:
/usr/include/c++/11/bits/ostream.tcc:119:5: note: candidate: ‘std::basic_ostream<_CharT, _Traits>& std::basic_ostream<_CharT, _Traits>::operator<<(std::basic_ostream<_CharT, _Traits>::__streambuf_type*) [with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT, _Traits>::__streambuf_type = std::basic_streambuf<char>]’
  119 |     basic_ostream<_CharT, _Traits>::
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/c++/11/bits/ostream.tcc:120:34: note:   no known conversion for argument 1 from ‘Temperature’ to ‘std::basic_ostream<char>::__streambuf_type*’ {aka ‘std::basic_streambuf<char>*’}
  120 |     operator<<(__streambuf_type* __sbin)
      |                ~~~~~~~~~~~~~~~~~~^~~~~~
In file included from /usr/include/c++/11/bits/basic_string.h:48,
                 from /usr/include/c++/11/string:55,
                 from /usr/include/c++/11/bits/locale_classes.h:40,
                 from /usr/include/c++/11/bits/ios_base.h:41,
                 from /usr/include/c++/11/ios:42,
                 from /usr/include/c++/11/ostream:38,
                 from /usr/include/c++/11/iostream:39,
                 from a.cpp:1:
/usr/include/c++/11/string_view:667:5: note: candidate: ‘template<class _CharT, class _Traits> std::basic_ostream<_CharT, _Traits>& std::operator<<(std::basic_ostream<_CharT, _Traits>&, std::basic_string_view<_CharT, _Traits>)’
  667 |     operator<<(basic_ostream<_CharT, _Traits>& __os,
      |     ^~~~~~~~
/usr/include/c++/11/string_view:667:5: note:   template argument deduction/substitution failed:
a.cpp:7:28: note:   ‘Temperature’ is not derived from ‘std::basic_string_view<_CharT, _Traits>’
    7 |   std::cout << Temperature{};
      |                            ^
In file included from /usr/include/c++/11/string:55,
                 from /usr/include/c++/11/bits/locale_classes.h:40,
                 from /usr/include/c++/11/bits/ios_base.h:41,
                 from /usr/include/c++/11/ios:42,
                 from /usr/include/c++/11/ostream:38,
```

Refactoring & type migration

- Existing approaches usually make you *design* the new type
- ...and *define* a mapping to execute.
- If the design does not cover, the perimeter of the migration will not compile.

Refactoring & type migration

- Existing approaches usually make you *design* the new type
- ...and *define* a mapping to execute.
- If the design does not cover, the perimeter of the migration will not compile.

Problem: Code does not compile $\Longrightarrow$ no more tooling support.

The big problem?

$$10^9 - 10^{11}$$

Richárd Szalay

The Rough Road Towards Upgrading to C++ Modules

Assume that you have already...

- Some existing software project
- A well-designed Strong Type library
- Preemptively existing mapping for type transition

Assume that you have already...

- Some existing software project
- A well-designed strong type library
- Preemptively existing mapping for type transition

Refactoring & type migration

Assume that you have already...

- Some existing software project
- A well-designed strong type library
- Preemptively existing mapping for type transition
- Some sort of tooling?

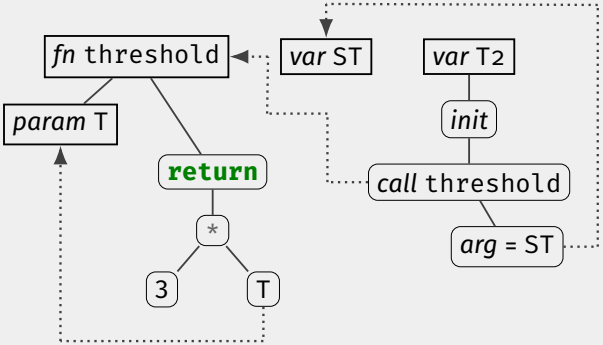
Assume that you have already...

- Some existing software project
- A well-designed strong type library
- Preemptively existing mapping for type transition
- Some sort of tooling?

Let's refactor!

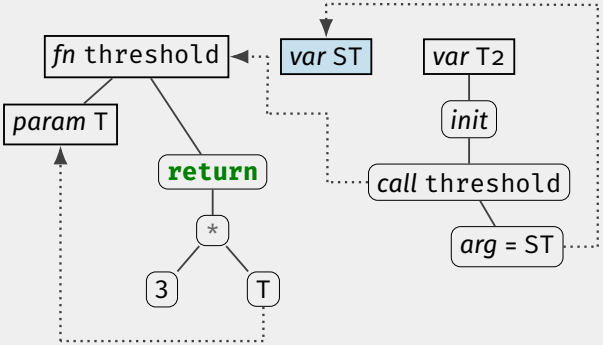
Step I: Seeding an initial taint

```
int threshold(int T)
{
    return 3 * T;
}

int ST = …;
int T2 = threshold(ST);
```



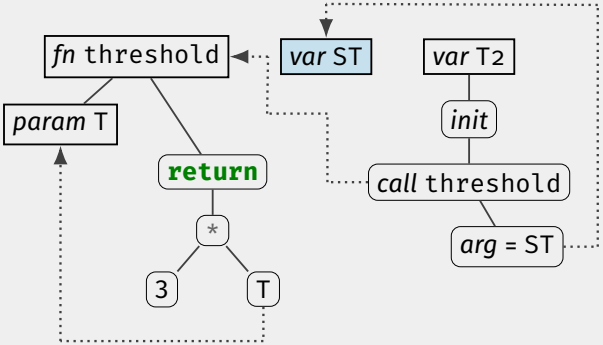
Step I: Seeding an initial taint

```
int threshold(int T)
{
    return 3 * T;
}

[[fictive_type(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



Step I: Seeding an initial taint

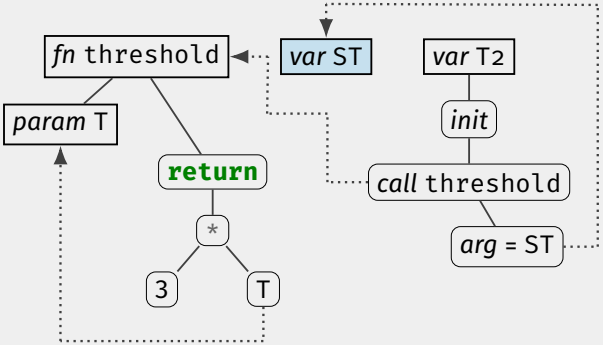
```
int threshold(int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



```
❯ g++ -c temperature.cpp
temperature.cpp: In function ‘int main()’:
temperature.cpp:5:39: warning: ‘fictive_type’ attribute directive ignored [-Wattributes]
    5 |   [[fictive_type("temperature")]] int ST = read_sensor();
      |                                       ^~
```

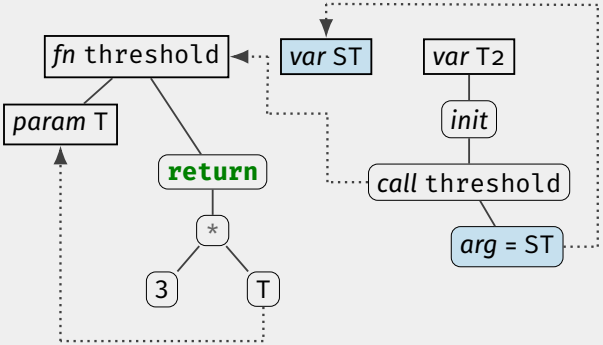
Step II: Initial conditions

```
int threshold(int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



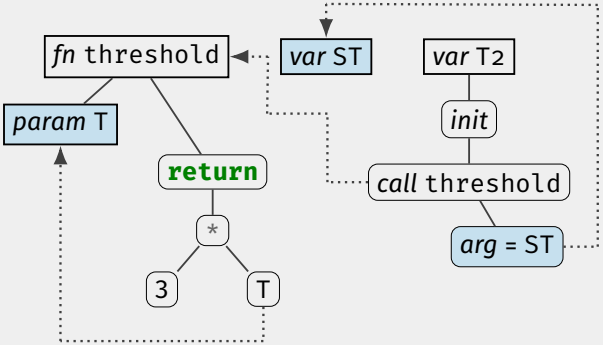
Step II: Propagation — Round 1

```
int threshold(int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



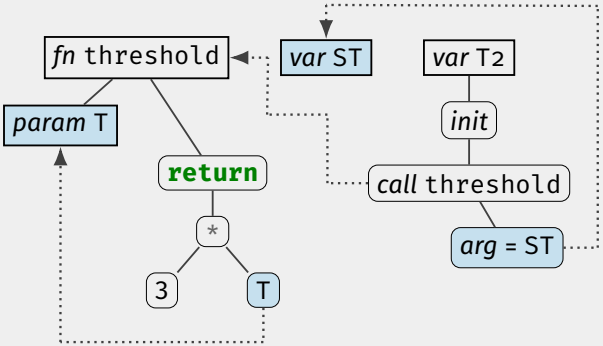
Step II: Propagation — Round 1

```
int threshold(
    [[ft(temperature)]] int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



Step II: Propagation — Round 1

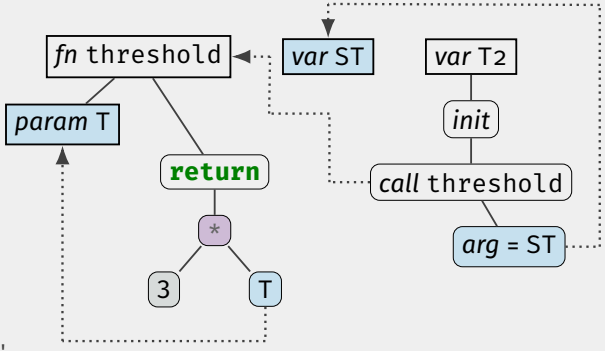
```
int threshold(
    [[ft(temperature)]] int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



Step II: Propagation — Round 1

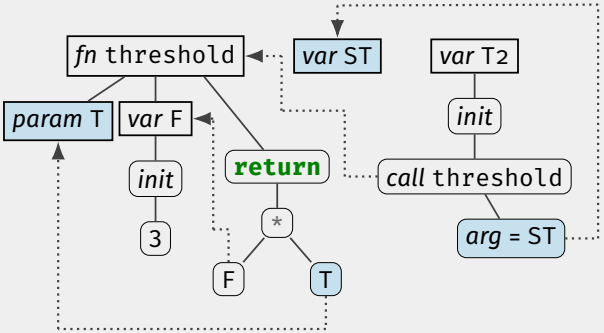
```
int threshold(
    [[ft(temperature)]] int T)
{
    return 3 * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```

```
test.cpp:1:32: warning: use of undefined
    fictive operator '<?> * temperature'

test.cpp:1:30: note: left operand is 'int'
    literal, configure overload or refactor
    into variable
```



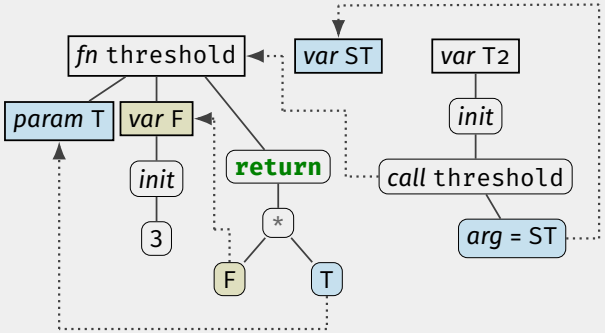
Step II: Propagation — Round 1 – Refactoring

```
int threshold(
    [[ft(temperature)]] int T)
{
    int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



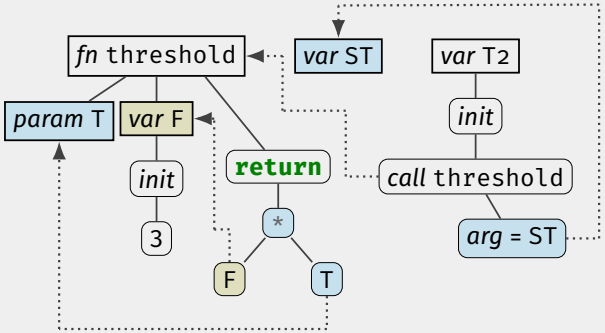
Step II: Propagation — Round 1 – User's changes

```
/* factor * temperature = temperature */

int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



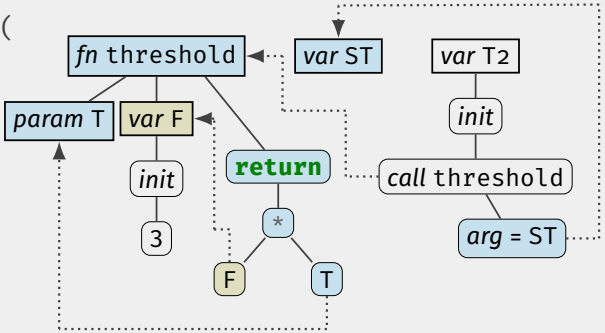
Step II: Propagation — Round 2

```
/* factor * temperature = temperature */

int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



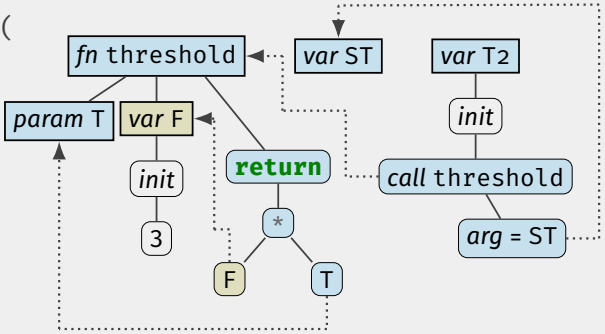
Step II: Propagation — Round 2

```
/* factor * temperature = temperature */

[[ft(temperature)]] int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
int T2 = threshold(ST);
```



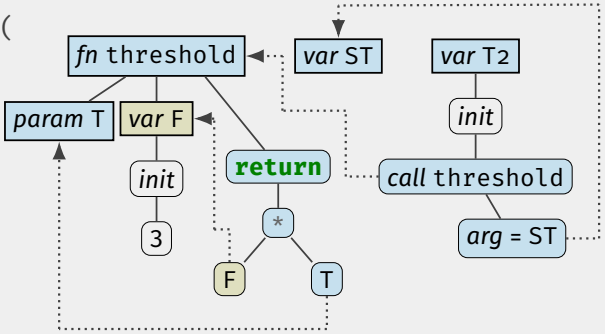
Step II: Propagation — Round 2

```
/* factor * temperature = temperature */

[[ft(temperature)]] int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
[[ft(temperature)]]
    int T2 = threshold(ST);
```



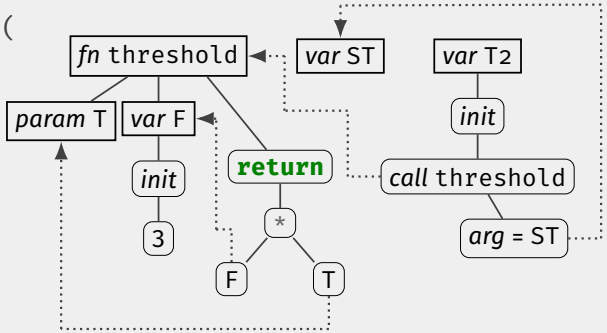
Fix point

```
/* factor * temperature = temperature */

[[ft(temperature)]] int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
[[ft(temperature)]]
    int T2 = threshold(ST);
```



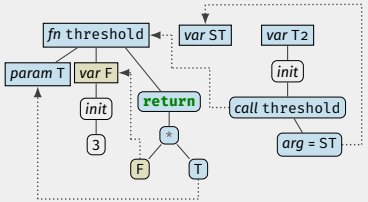
Still perfectly valid C++

```
int threshold(
    int T)
{
    int F = 3;
    return F * T;
}

int ST = …;

int T2 = threshold(ST);
```



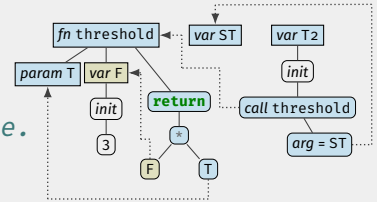
Step IIIa: Code generation

```
class temperature { /* … */ };
class factor      { /* … */ };

[[ft(temperature)]] int threshold(
    [[ft(temperature)]] int T)
{
    [[ft(factor)]] int F = 3;
    return F * T;
}

[[ft(temperature)]]
    int ST = …;
[[ft(temperature)]]
    int T2 = threshold(ST);
```



Step IIIb: Refactoring

```
class temperature { /* … */ };
class factor      { /* … */ };

temperature threshold(temperature T)
{
    factor F{3}; // Explicit cast to strong type.
    return F * T;
}

temperature ST = …;
temperature T2 = threshold(ST);
```



Step III: Code generation & refactoring

```
class temperature { /* … */ };
class factor      { /* … */ };

temperature threshold(temperature T)
{
    factor F{3}; // Explicit cast to strong type.
    return F * T;
}

temperature ST = …;
temperature T2 = threshold(ST);
```

```
test.cpp:11:14: error: invalid operands to binary
    '*' expression ('factor' and 'temperature')
         return F * T;
                ~ ^ ~
```

Step III: Code generation & refactoring

```
class temperature { /* … */ };
class factor      { /* … */ };
temperature operator *(factor F, temperature T) { /* … */ }

temperature threshold(temperature T)
{
    factor F{3}; // Explicit cast to strong type.
    return F * T;
}

temperature ST = …;
temperature T2 = threshold(ST);
```

Richárd Szalay and Zoltán Porkoláb. **"Flexible semi-automatic support for type migration of primitives for C/C++ programs".** In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Mar. 15, 2022, pp. 878–889. DOI: 10.1109/SANER53432.2022.00106. URL: http://ieeexplore.ieee.org/document/9825812

1 Introduction

2 Overview

3 In detail
- What is a “fictive type”?
  Side note: Taint analysis
- Code generation
- ~~Problems~~ Design decisions
  Ignores
  Overloads
  Inlines and summaries

4 Tooling & “Demo”

1 Introduction

2 Overview

3 In detail
- What is a “fictive type”?
  Side note: Taint analysis
- Code generation
- ~~Problems~~ Design decisions
  - Ignores
  - Overloads
  - Inlines and summaries

4 Tooling & “Demo”

[[fictive_type("T")]]

- Transitional fuel
- In-band signal (mostly…)
- Strict "is-a": T *is-a* `temperature`

```
[[fictive_type("temperature")]] int T
[[ft("temperature")]] int T
```

[[fictive_type("T")]]

- Transitional fuel
- In-band signal (mostly…)
- Strict "is-a": T *is-a* `temperature`

```
[[fictive_type("temperature")]] int T
[[ft("temperature")]] int T
```

```
[[fictive_type("temperature", "velocity")]] int T
```

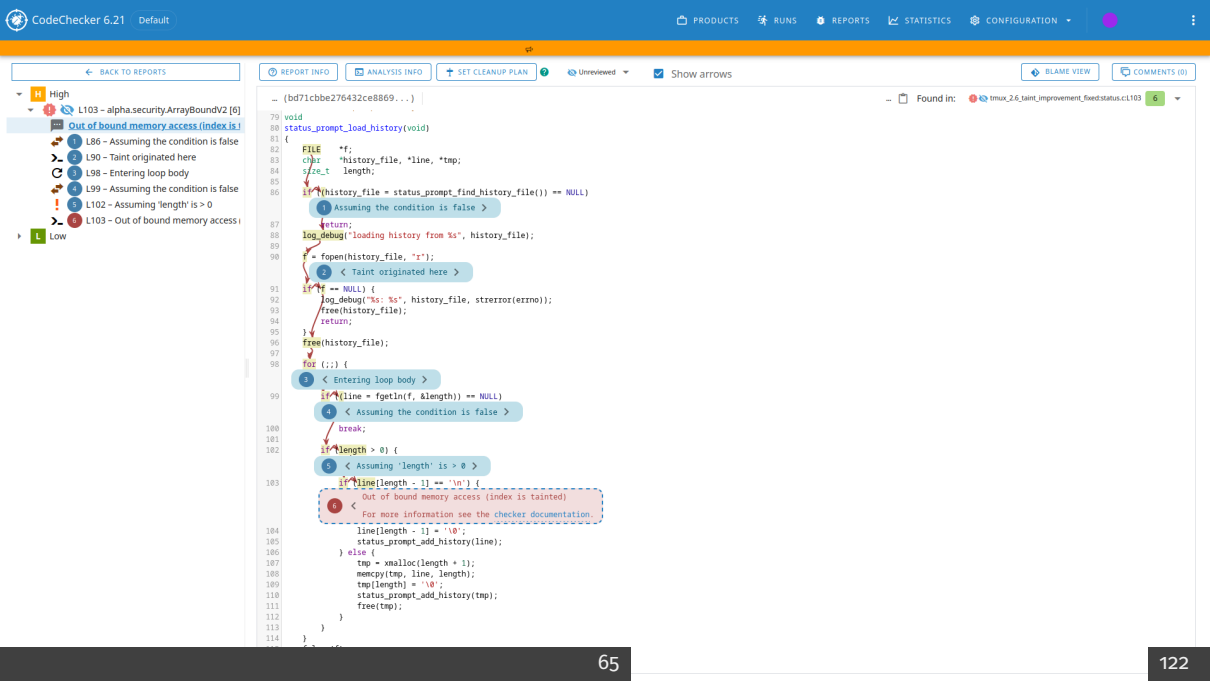
[[fictive_type("T")]]

- Transitional fuel
- In-band signal (mostly…)
- Strict "is-a": T *is-a* `temperature`

```
[[fictive_type("temperature")]] int T
[[ft("temperature")]] int T
```

```
[[fictive_type("temperature", "velocity")]] int T
```

```
[[fictive_type("temperature"), fictive_type("velocity")]] int T
```


CodeChecker 6.21
Default
PRODUCTS
RUNS
REPORTS
STATISTICS
CONFIGURATION
BACK TO REPORTS
REPORT INFO
ANALYSIS INFO
SET CLEANUP PLAN
Unreviewed
Show arrows
BLAME VIEW
COMMENTS (0)
High
L103 – alpha.security.ArrayBoundV2 [6]
Out of bound memory access (index is t
L86 – Assuming the condition is false
L90 – Taint originated here
L98 – Entering loop body
L99 – Assuming the condition is false
L102 – Assuming 'length' is > 0
L103 – Out of bound memory access
Low
{bd71cbbe276432ce8869...}
Found in:
tmux_2.6_taint_improvement_fixed:status.c:L103
6
Assuming the condition is false
Taint originated here
Entering loop body
Assuming the condition is false
Assuming 'length' is > 0
Out of bound memory access (index is tainted)
For more information see the checker documentation.

Taint analysis

- Security-first static analysis technique
- Usually path-, flow-, and/or context sensitive
- Performed on data-flow, generally a “must all-path” problem
- Interested in the **value**

Side note: Taint analysis

Taint analysis

- Security-first static analysis technique
- Usually path-, flow-, and/or context sensitive
- Performed on data-flow, generally a “must all-path” problem
- Interested in the **value**

Fictive Types

- Design-first / comprehension-first static analysis technique
- Path-, and context-**in**sensitive, locally flow-sensitive
- Strictly a “must all-path” problem
- Interested in the **type**
- (Not entirely security-first, but *could* be adapted?)

Side note: Taint analysis

```
std::ostream& std::ostream::operator>>(
    std::ostream&, [[fictive_type("maybe taint")]] char(&)[]);
std::FILE* std::fopen(
    [[fictive_type("must-not taint")]] const char*, const char*);


int main()
{

      char Buf[PATH_MAX];
  std::cin >> Buf;
  std::fopen(Buf, "r");
}
```

Side note: Taint analysis – Reverse propagation

```
std::ostream& std::ostream::operator>>(
    std::ostream&, [[fictive_type("maybe taint")]] char(&)[] );
std::FILE* std::fopen(
    [[fictive_type("must-not taint")]] const char*, const char*);


int main()
{
  [[fictive_type("maybe taint")]]
      char Buf[PATH_MAX];
  std::cin >> Buf;
  std::fopen(Buf, "r");
}
```

Side note: Taint analysis – Forward propagation

```
std::ostream& std::ostream::operator>>(
    std::ostream&, [[fictive_type("maybe taint")]] char(&)[]);
std::FILE* std::fopen(
    [[fictive_type("must-not taint")]] const char*, const char*);


int main()
{
  [[fictive_type("maybe taint")]]
      char Buf[PATH_MAX];
  std::cin >> Buf;
  std::fopen( Buf , "r");
}
```

Side note: Taint analysis – Trigger error

```
std::FILE* std::fopen(
    [[fictive_type("must-not taint")]]
        ↕ ↯ disjoint FT collision ↕
    [[fictive_type("maybe taint")]] const char*,
    const char*);

[[fictive_type("maybe taint")]] char Buf[PATH_MAX];
std::fopen( Buf , "r");
```

[[fictive_type("T")]]

```
int fn([[fictive_type("T")]] int P);
struct S { [[fictive_type("count")]] int Size; };
```

[[fictive_type("T")]]

```
int fn([[fictive_type("T")]] int P);
struct S { [[fictive_type("count")]] int Size; };
```

```
[[ft("count")]] 5L
int int::operator+([[ft("T")]] int L, int R);
```

[[fictive_type("T")]]

```
int fn([[fictive_type("T")]] int P);
struct S { [[fictive_type("count")]] int Size; };
```

```
[[ft("count")]] 5L
int int::operator+([[ft("T")]] int L, int R);
```

OperationMap.yaml

```yaml
BuiltinOperations:
  Binary:
    - Operation: "temperature - temperature"
      Result: "temperature_difference_t"
    - Operation: "time * acceleration"
      Result: "velocity"
      Commutative: true
```

Code generation

```yaml
StrongTypes:
  - TypeName: "Distance"
    Wraps: "float"
    BinaryOperations:
      - Operation: "/"
        ArgType: "Time"
        ResType: "Speed"
      - Operation: "+"
        ArgType: "Distance"
        ResType: "Distance"
      - Operation: "+="
        ArgType: "Distance"
        ResType: "Distance"
  - TypeName: "Time"
    Wraps: "float"
    BinaryOperations:
      - Operation: "+"
        ArgType: "Time"
        ResType: "Time"
      - Operation: "+="
        ArgType: "Time"
        ResType: "Time"
  - TypeName: "Speed"
    Wraps: "float"

# ...
```

Code generation

```
root@532a7e7c74a2:/src/sys_tests/AverageSpeed# \
    /usr/bin/STGenerator ./StrongTypes.yaml
YAML configuration satisfies all validation rules.
Strong types library was generated.
Update CMakeLists.txt to add library to the project:

```

```
    add_subdirectory(StrongTypes)
    target_link_libraries(<target_name> PUBLIC StrongTypes)
```

Code generation

```
#include "Speed.h"
#include "Time.h"
class Distance {
  float value;
public:
  using wrapped_type = float;

  explicit constexpr Distance(
    const wrapped_type& V) : value(V) {}
  explicit operator wrapped_type() const
    { return value; }
  constexpr wrapped_type& get()
    { return value; }
  constexpr wrapped_type  get() const
    { return value; }

  inline Speed     operator/ (const Time& rhs) const;
  inline Distance  operator+ (const Distance& rhs) const;
  inline Distance& operator+=(const Distance& rhs);
};
```

```
Speed Distance::operator/(const Time& rhs)
    const {
  return Speed{ value /
    static_cast<Time::wrapped_type>(rhs) };
}
Distance Distance::operator+(
    const Distance& rhs) const {
  return Distance{ value +
    static_cast<wrapped_type>(rhs) };
}
Distance& Distance::operator+=(
    const Distance& rhs) {
  value += static_cast<wrapped_type>(rhs);
  return *this;
}
```

1 Introduction

2 Overview

3 In detail
- What is a “fictive type”?
  Side note: Taint analysis
- Code generation
- ~~Problems~~ Design decisions
  - Ignores
  - Overloads
  - Inlines and summaries

4 Tooling & “Demo”

Where does this all go wrong?

Where does this all go wrong?

Unrefactorables

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)
- C API inter-op (if custom invariants are needed)

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)
- C API inter-op (if custom invariants are needed)
- “Very generic” generics

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)
- C API inter-op (if custom invariants are needed)
- “Very generic” generics

How to control requirements?

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)
- C API inter-op (if custom invariants are needed)
- “Very generic” generics

How to control requirements?

- (Project-)inner perimeters

Where does this all go wrong?

Unrefactorables

- All & every library (STL, POSIX, WinAPI, other 3rd-party)
- C API inter-op (if custom invariants are needed)
- “Very generic” generics

How to control requirements?

- (Project-)inner perimeters
- Combat over-approximation

Problem

You **can not** and **must not** rewrite external, API headers.

```
std::ostream& operator<<(std::ostream&, int);
int errno;
void std::vector<T>::push_back(const T&);
```

API boundaries & controlling the perimeter

[[fictive_type_ignore]]

Mark entity as a “black hole” in the FT-domain.

```
[[ft_ignore]] std::ostream& operator<<(std::ostream&, int);
[[ft_ignore]] int errno;
[[ft_ignore]] void std::vector<T>::push_back(const T&); // Some additional questions...
```

```
[[ft(”temperature”)]] int T;
std::cout << T;
```

```
temperature T;
std::cout << static_cast<int>(T);
```

[[fictive_type_barrier]]

Marks an entity to be the terminus for refactoring – but allow its refactoring!

```
[[ft_barrier]] int PseudoSink;
```

```
[[ft("temperature")]] int T;
[[ft_barrier]] int T2 = T;
int T3 = T2;
```

```
temperature T;
temperature T2 = T;
int T3 = static_cast<int>(T2);
```

Problem

`[[ft_ignore]]` may generate verbose and ugly casts at every call.

```
std::ostream& operator<<(std::ostream&, int);
std::cout << static_cast<int>(T) << static_cast<double>(V);
```

Overload generation

[[fictive_type_overload]]

Create an overload of the refactored function and *unbox* the stronger type inside.

```
std::ostream& operator<<(std::ostream&, [[ft_overload]] int);
```

```
[[ft("temperature")]] int T;
std::cout << T;
```

```
std::ostream& operator<<(
    std::ostream& LHS,
    const temperature& RHS) {
  return (LHS << static_cast<int>(T));
}

temperature T;
std::cout << T;
```

Problem

Several families of functions where:

- refactoring is not the right option
- ignoring and evacuating the FT-domain is not the right option

```
template <typename T> T max(T&&, T&&);
int max(int, int);
```

```
[[ft("temperature")]] int T1, T2;
int Tmax = max(T1, T2);
```

Problem

T1 and T2 hit the params, they get tainted and refactored. Return type is lost.

```
// Where did 'int max(int, int);' go?
int max(temperature, temperature);
```

```
temperature T1, T2;
int Tmax = max(T1, T2);
```

Ignorance is a problem

Problem

Evacuating the FT-domain inserts **cast**s. Return type is lost, again...

```
[[ft_ignore]] int max(int, int);

temperature T1, T2;
int Tmax = max(static_cast<int>(T1), static_cast<int>(T2));
```

[[fictive_type_inline]]

Flow fictive type through the function transparently, if possible.

```
[[ft_inline]] int max(int a, int b) { return (a < b) ? b : a; }
// Summary: "return_type(max) == type(a) == type(b)"
```

```
int max(int, int) { /* ... */ }

[[ft("temperature")]] int T1, T2;
int Tmax = max(T1, T2);
```

```
int max(int, int) { /* ... */ }

temperature T1, T2;
temperature Tmax{max(
  static_cast<int>(T1),
  static_cast<int>(T2)
)};
```

1 Introduction

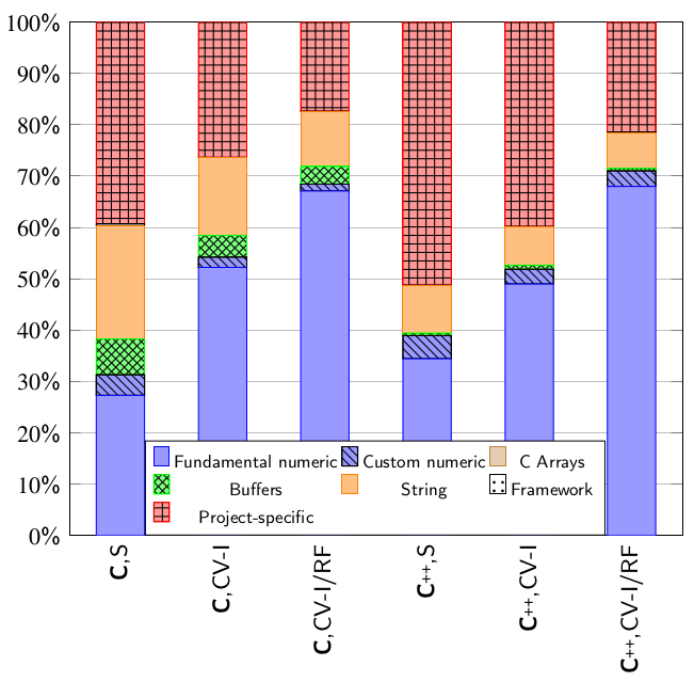
2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

The Timeline

- Late 2018 - Early 2019: Research into parameter passing and unsafe interfaces
- 2020 Feb.: Foundation stone laying ceremony, first commit for *Fictive Types* modelling in Clang
- 2020 Jul.: In-band signalling not only read, but created
- 2021 Jul.: Operation map implemented for strengthening operators over built-ins
- **2021 Dec. 9**: C-level tool works as fix-point iteration, but does not scale


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Fundamental numeric
Custom numeric
C Arrays
Buffers
String
Framework
Project-specific
C,S
C,CV-I
C,CV-I/RF
C++,S
C++,CV-I
C++,CV-I/RF

The Timeline

- Late 2018 - Early 2019: Research into parameter passing and unsafe interfaces
- 2020 Feb.: Foundation stone laying ceremony, first commit for *Fictive Types* modelling in Clang
- 2020 Jul.: In-band signalling not only read, but created
- 2021 Jul.: Operation map implemented for strengthening operators over built-ins
- **2021 Dec. 9**: C-level tool works as fix-point iteration, but does not scale
- Christmas & New Year 2021→2022: Implement custom data structure to make the tool scalable
- 2022 Mar.: Serialisation (`-collect`) prototype works
- 2022 Spring (semester): Work on generating the *strong interface* with a colleague from ELTE
- 2022 Sep.: Work on refactoring and re-implementing the operation map in the new ecosystem

Inputs

- Compilation database (CMake or `CodeChecker log`)

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!
- Configuration files (e.g., operation map) – can be versioned!

Fictive Type ecosystem

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!
- Configuration files (e.g., operation map) – can be versioned!

Tools

- `clang-fictive-types [compilation db]`: Overall driver

Fictive Type ecosystem

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!
- Configuration files (e.g., operation map) – can be versioned!

Tools

- `clang-fictive-types [compilation db]`: Overall driver
- `clang-fictive-types-collect`: Pre-analysis synthesis (aka. “CTU-collect”)

Fictive Type ecosystem

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!
- Configuration files (e.g., operation map) – can be versioned!

Tools

- `clang-fictive-types [compilation db]`: Overall driver
- `clang-fictive-types-collect`: Pre-analysis synthesis (aka. “CTU-collect”)
- `clang-fictive-types-flow`: Fix-point iteration

Fictive Type ecosystem

Inputs

- Compilation database (CMake or `CodeChecker log`)
- Generated code must have been generated
- The initial taint (in-band signal!) – can be versioned!
- Configuration files (e.g., operation map) – can be versioned!

Tools

- `clang-fictive-types [compilation db]`: Overall driver
- `clang-fictive-types-collect`: Pre-analysis synthesis (aka. “CTU-collect”)
- `clang-fictive-types-flow`: Fix-point iteration
- `clang-fictive-types-diagnose`: Code generation and rewriting

1 Introduction

2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

Demo setup

header.h

```
namespace lib {
  extern int DangerousGlobalVariable;
  int calc(int I);
}
```

main.cpp

```
#include "header.h"

int main(int ArgC, char* ArgV[]) {
  int X = lib::calc(ArgC);
  int X2 = X + X + X;

  return X2;
}
```

lib.cpp

```
#include "header.h"

namespace lib {
  int DangerousGlobalVariable = 42;
  int calc(int I) {
    DangerousGlobalVariable += I;
    return I;
  }
}
```

```
→  fictive-types-test
```

```
➜  fictive-types-test ll
.rw-r--r--  90 13 ápr   14:31 H header.h
.rw-r--r-- 150 13 ápr   13:25 C lib.cpp
.rw-r--r-- 127 13 ápr   13:25 C main.cpp
➜  fictive-types-test
```

```
➜  fictive-types-test ll
.rw-r--r--  90 13 ápr   14:31 H header.h
.rw-r--r-- 150 13 ápr   13:25 C lib.cpp
.rw-r--r-- 127 13 ápr   13:25 C main.cpp
➜  fictive-types-test CodeChecker log -b "g++ -c lib.cpp; g++ -c main.cpp; g++ main.o lib.o -o prog" -o compile_commands.json
[INFO 2023-04-13 14:58] - Starting build...
[INFO 2023-04-13 14:58] - Using CodeChecker ld-logger.
[INFO 2023-04-13 14:58] - Build finished successfully.
➜  fictive-types-test
```

```
➜  fictive-types-test ll
.rw-r--r--  90 13 ápr   14:31 H header.h
.rw-r--r-- 150 13 ápr   13:25  lib.cpp
.rw-r--r-- 127 13 ápr   13:25  main.cpp
➜  fictive-types-test CodeChecker log -b "g++ -c lib.cpp; g++ -c main.cpp; g++ main.o lib.o -o prog" -o compile_commands.json
[INFO 2023-04-13 14:58] - Starting build...
[INFO 2023-04-13 14:58] - Using CodeChecker ld-logger.
[INFO 2023-04-13 14:58] - Build finished successfully.
➜  fictive-types-test cat compile_commands.json
```

File: **compile_commands.json**

```
[
    {
        "directory": "/local/repo/llvm-project/fictive-types-test",
        "command": "/usr/bin/g++ -c lib.cpp",
        "file": "lib.cpp"
    }
    ,
    {
        "directory": "/local/repo/llvm-project/fictive-types-test",
        "command": "/usr/bin/g++ -c main.cpp",
        "file": "main.cpp"
    }
]
```

```
➜  fictive-types-test
```

```
➜  fictive-types-test ll
.rw-r--r--  90 13 ápr   14:31 H header.h
.rw-r--r-- 150 13 ápr   13:25  lib.cpp
.rw-r--r-- 127 13 ápr   13:25  main.cpp
➜  fictive-types-test CodeChecker log -b "g++ -c lib.cpp; g++ -c main.cpp; g++ main.o lib.o -o prog" -o compile_commands.json
[INFO 2023-04-13 14:58] - Starting build...
[INFO 2023-04-13 14:58] - Using CodeChecker ld-logger.
[INFO 2023-04-13 14:58] - Build finished successfully.
➜  fictive-types-test cat compile_commands.json
```

File: **compile_commands.json**

```
[
    {
        "directory": "/local/repo/llvm-project/fictive-types-test",
        "command": "/usr/bin/g++ -c lib.cpp",
        "file": "lib.cpp"
    }
    ,
    {
        "directory": "/local/repo/llvm-project/fictive-types-test",
        "command": "/usr/bin/g++ -c main.cpp",
        "file": "main.cpp"
    }
]
```

```
➜  fictive-types-test ./prog 1 2 3 4 5; echo $?; echo $(( 3 * 6 ))
18
18
➜  fictive-types-test
```

```
➜  fictive-types-test clang-fictive-types ./compile_commands.json --data-dir ./tmp --jobs 1
[1/2] Collecting AST for lib.cpp
Build AST begin Thu Apr 13 14:59:39 2023
Build AST end   Thu Apr 13 14:59:39 2023
Collecting source translation unit 'lib.cpp'...
Visit AST begin Thu Apr 13 14:59:39 2023
Visit AST end   Thu Apr 13 14:59:39 2023
Writing Propagation data structure to output file './tmp/tu-lib.cpp.dat'.
Write Data begin Thu Apr 13 14:59:39 2023
. ChunkFile Version 1
|-- 0x55db26ad7c30 Container '<root>' (id-len: 6) (Writable)
|   |-- 0x55db26ae88c0 Container '<meta>' (id-len: 6) (Writable)
|   |   |-- 0x55db26ad9f50 Payload 0x01 'DataType' (id-len: 8) (Writable) WrittenBytes: 20
|   |   |-- 0x55db26ad32a0 Payload 0x01 'MainFile' (id-len: 8) (Writable) WrittenBytes: 12
|   |   |-- 0x55db26addde0 Payload 0x02 'DeclClasses' (id-len: 11) (Writable) WrittenBytes: 44
|   |   |-- 0x55db26acbf50 Payload 0x02 'StmtClasses' (id-len: 11) (Writable) WrittenBytes: 66
|   |-- 0x55db26ad1a80 Container '<string-tables>' (id-len: 15) (Writable)
|   |   |-- 0x55db26accb20 Strings
|   |   |   |-- 0x55db26accb20 Vector 0xFFFF 'Filepaths' (id-len: 9) NumElements = 2 (Writable) WrittenBytes: 112
|   |   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x55db26ad3490 Strings
|   |   |   |-- 0x55db26ad3490 Vector 0xFFFF 'Typenames' (id-len: 9) NumElements = 0 (Writable) WrittenBytes: 0
|   |   |   |   |-- LookUpTable Size = 0
|   |-- 0x55db26accab0 Container '<node-vectors>' (id-len: 14) (Writable)
|   |   |-- 0x55db26b53ac0 Vector 0x04 'Decls' (id-len: 5) NumElements = 3 (Writable) WrittenBytes: 324
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x55db26b5c450 Vector 0x08 'Stmts' (id-len: 5) NumElements = 3 (Writable) WrittenBytes: 236
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x55db26ad3630 Vector 0x0C 'Locations&Ranges' (id-len: 16) NumElements = 1 (Writable) WrittenBytes: 28
|   |   |   |-- LookUpTable Size = 1
|   |-- 0x55db26acc640 Payload 0x10 'Propagations' (id-len: 12) (Writable) WrittenBytes: 68
Write Data end   Thu Apr 13 14:59:39 2023
[2/2] Collecting AST for main.cpp
Build AST begin Thu Apr 13 14:59:40 2023
Build AST end   Thu Apr 13 14:59:40 2023
Collecting source translation unit 'main.cpp'...
Visit AST begin Thu Apr 13 14:59:40 2023
Visit AST end   Thu Apr 13 14:59:40 2023
Writing Propagation data structure to output file './tmp/tu-main.cpp.dat'.
Write Data begin Thu Apr 13 14:59:40 2023
```

```
Visit AST begin Thu Apr 13 14:59:40 2023
Visit AST end   Thu Apr 13 14:59:40 2023
Writing Propagation data structure to output file './tmp/tu-main.cpp.dat'.
Write Data begin Thu Apr 13 14:59:40 2023
. ChunkFile Version 1
|-- 0x562a76bd0c40 Container '<root>' (id-len: 6) (Writable)
|   |-- 0x562a76be18d0 Container '<meta>' (id-len: 6) (Writable)
|   |   |-- 0x562a76c43660 Payload 0x01 'DataType' (id-len: 8) (Writable) WrittenBytes: 20
|   |   |-- 0x562a76bd8160 Payload 0x01 'MainFile' (id-len: 8) (Writable) WrittenBytes: 12
|   |   |-- 0x562a76bd6dc0 Payload 0x02 'DeclClasses' (id-len: 11) (Writable) WrittenBytes: 28
|   |   |-- 0x562a76bd8830 Payload 0x02 'StmtClasses' (id-len: 11) (Writable) WrittenBytes: 58
|   |-- 0x562a76bd1630 Container '<string-tables>' (id-len: 15) (Writable)
|   |   |-- 0x562a76bccf70 Strings
|   |   |   |-- 0x562a76bccf70 Vector 0xFFFF 'Filepaths' (id-len: 9) NumElements = 1 (Writable) WrittenBytes: 56
|   |   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x562a76bcd160 Strings
|   |   |   |-- 0x562a76bcd160 Vector 0xFFFF 'Typenames' (id-len: 9) NumElements = 0 (Writable) WrittenBytes: 0
|   |   |   |   |-- LookUpTable Size = 0
|   |-- 0x562a76bc5ab0 Container '<node-vectors>' (id-len: 14) (Writable)
|   |   |-- 0x562a76bc5b20 Vector 0x04 'Decls' (id-len: 5) NumElements = 2 (Writable) WrittenBytes: 184
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x562a76c4cad0 Vector 0x08 'Stmts' (id-len: 5) NumElements = 5 (Writable) WrittenBytes: 396
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x562a76bcd300 Vector 0x0C 'Locations&Ranges' (id-len: 16) NumElements = 1 (Writable) WrittenBytes: 28
|   |   |   |-- LookUpTable Size = 1
|   |-- 0x562a76bd2e50 Payload 0x10 'Propagations' (id-len: 12) (Writable) WrittenBytes: 101
Write Data end   Thu Apr 13 14:59:40 2023

=================== Begin iteration        1 ====================
[T1/1] Working on lib.cpp
[T1/1] Working on main.cpp
[T1/1] No files remaining. Shutting down...
Iteration begun Thu Apr 13 14:59:40 2023
Iteration ended Thu Apr 13 14:59:40 2023
Iteration duration 0 sec

===================  End  iteration        1 ====================

******************** At the beginning of iteration        2, no files are changed. ********************
➜  fictive-types-test
```

```
➜ fictive-types-test cat main.cpp
```

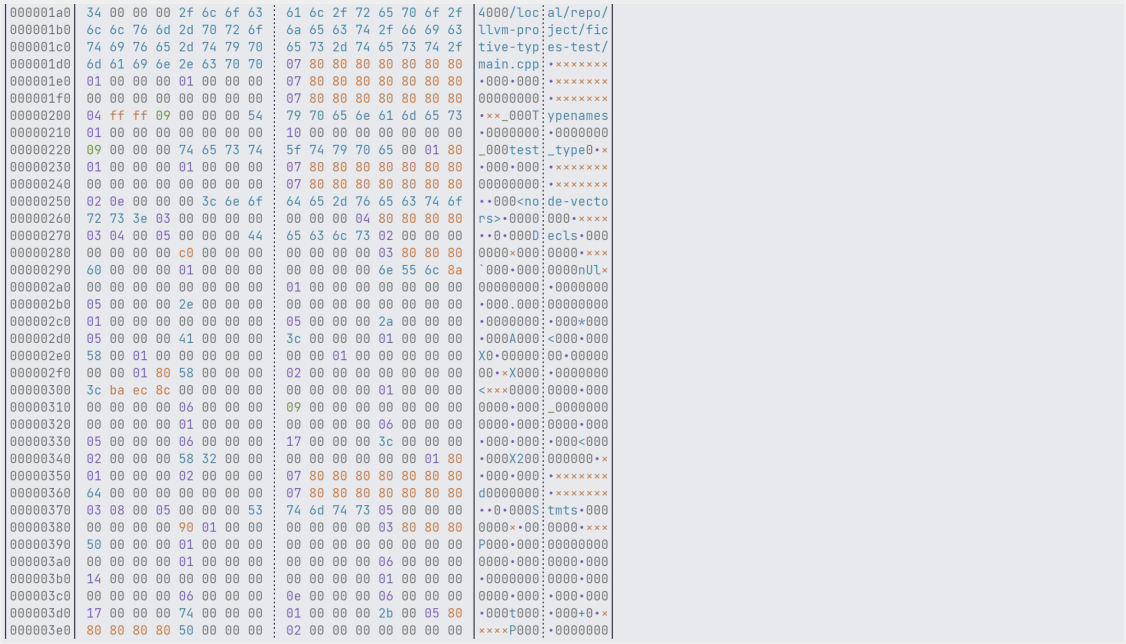
File: **main.cpp**

```cpp
#include "header.h"

int main(int ArgC, char* ArgV[])
{
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
    int X2 = X + X + X;

    return X2;
}
```

```
➜ fictive-types-test
```

```
    int X2 = X + X + X;

    return X2;
}
```

```
➜  fictive-types-test clang-fictive-types-collect -p . --output ./tmp/tu-main.cpp.dat main.cpp
Build AST begin Thu Apr 13 16:06:41 2023
Build AST end   Thu Apr 13 16:06:41 2023
Collecting source translation unit 'main.cpp'...
Visit AST begin Thu Apr 13 16:06:41 2023
Visit AST end   Thu Apr 13 16:06:41 2023
Writing Propagation data structure to output file './tmp/tu-main.cpp.dat'.
Write Data begin Thu Apr 13 16:06:41 2023
. ChunkFile Version 1
|-- 0x560c66478b80 Container '<root>' (id-len: 6) (Writable)
|   |-- 0x560c66489810 Container '<meta>' (id-len: 6) (Writable)
|   |   |-- 0x560c6647adf0 Payload 0x01 'DataType' (id-len: 8) (Writable) WrittenBytes: 20
|   |   |-- 0x560c664800a0 Payload 0x01 'MainFile' (id-len: 8) (Writable) WrittenBytes: 12
|   |   |-- 0x560c6647ed00 Payload 0x02 'DeclClasses' (id-len: 11) (Writable) WrittenBytes: 28
|   |   |-- 0x560c664eae10 Payload 0x02 'StmtClasses' (id-len: 11) (Writable) WrittenBytes: 58
|   |-- 0x560c66479570 Container '<string-tables>' (id-len: 15) (Writable)
|   |   |-- 0x560c66474eb0 Strings
|   |   |   |-- 0x560c66474eb0 Vector 0xFFFF 'Filepaths' (id-len: 9) NumElements = 1 (Writable) WrittenBytes: 56
|   |   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x560c664750a0 Strings
|   |   |   |-- 0x560c664750a0 Vector 0xFFFF 'Typenames' (id-len: 9) NumElements = 1 (Writable) WrittenBytes: 14
|   |   |   |   |-- LookUpTable Size = 1
|   |-- 0x560c6646d9f0 Container '<node-vectors>' (id-len: 14) (Writable)
|   |   |-- 0x560c6646da60 Vector 0x04 'Decls' (id-len: 5) NumElements = 2 (Writable) WrittenBytes: 192
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x560c664f4a10 Vector 0x08 'Stmts' (id-len: 5) NumElements = 5 (Writable) WrittenBytes: 396
|   |   |   |-- LookUpTable Size = 1
|   |   |-- 0x560c66475240 Vector 0x0C 'Locations&Ranges' (id-len: 16) NumElements = 1 (Writable) WrittenBytes: 28
|   |   |   |-- LookUpTable Size = 1
|   |-- 0x560c66466aa0 Payload 0x10 'Propagations' (id-len: 12) (Writable) WrittenBytes: 101
Write Data end   Thu Apr 13 16:06:41 2023
➜  fictive-types-test
```



```
000001a0  34 00 00 00 2f 6c 6f 63  61 6c 2f 72 65 70 6f 2f  4000/loc al/repo/
000001b0  6c 6c 76 6d 2d 70 72 6f  6a 65 63 74 2f 66 69 63  llvm-pro ject/fic
000001c0  74 69 76 65 2d 74 79 70  65 73 2d 74 65 73 74 2f  tive-typ es-test/
000001d0  6d 61 69 6e 2e 63 70 70  07 80 80 80 80 80 80 80  main.cpp ·×××××××
000001e0  01 00 00 00 01 00 00 00  07 80 80 80 80 80 80 80  ·000·000 ·×××××××
000001f0  00 00 00 00 00 00 00 00  07 80 80 80 80 80 80 80  00000000 ·×××××××
00000200  04 ff ff 09 00 00 00 54  79 70 65 6e 61 6d 65 73  ·××_000T ypenames
00000210  01 00 00 00 00 00 00 00  10 00 00 00 00 00 00 00  ·0000000 ·0000000
00000220  09 00 00 00 74 65 73 74  5f 74 79 70 65 00 01 80  _000test _type0·×
00000230  01 00 00 00 01 00 00 00  07 80 80 80 80 80 80 80  ·000·000 ·×××××××
00000240  00 00 00 00 00 00 00 00  07 80 80 80 80 80 80 80  00000000 ·×××××××
00000250  02 0e 00 00 00 3c 6e 6f  64 65 2d 76 65 63 74 6f  ··000<no de-vecto
00000260  72 73 3e 03 00 00 00 00  00 00 00 04 80 80 80 80  rs>·0000 000·××××
00000270  03 04 00 05 00 00 00 44  65 63 6c 73 02 00 00 00  ··0·000D ecls·000
00000280  00 00 00 00 c0 00 00 00  00 00 00 00 03 80 80 80  0000×000 0000·×××
00000290  60 00 00 00 01 00 00 00  00 00 00 00 6e 55 6c 8a  `000·000 0000nUl×
000002a0  00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  00000000 ·0000000
000002b0  05 00 00 00 2e 00 00 00  00 00 00 00 00 00 00 00  ·000.000 00000000
000002c0  01 00 00 00 00 00 00 00  05 00 00 00 2a 00 00 00  ·0000000 ·000*000
000002d0  05 00 00 00 41 00 00 00  3c 00 00 00 01 00 00 00  ·000A000 <000·000
000002e0  58 00 01 00 00 00 00 00  00 00 01 00 00 00 00 00  X0·00000 00·00000
000002f0  00 00 01 80 58 00 00 00  02 00 00 00 00 00 00 00  00··×X00 ·0000000
00000300  3c ba ec 8c 00 00 00 00  00 00 00 00 01 00 00 00  <×××0000 0000·000
00000310  00 00 00 00 06 00 00 00  09 00 00 00 00 00 00 00  0000·000 _0000000
00000320  00 00 00 00 01 00 00 00  00 00 00 00 06 00 00 00  0000·000 0000·000
00000330  05 00 00 00 06 00 00 00  17 00 00 00 3c 00 00 00  ·000·000 ·000<000
00000340  02 00 00 00 58 32 00 00  00 00 00 00 00 00 01 80  ·000X200 000000·×
00000350  01 00 00 00 02 00 00 00  07 80 80 80 80 80 80 80  ·000·000 ·×××××××
00000360  64 00 00 00 00 00 00 00  07 80 80 80 80 80 80 80  d0000000 ·×××××××
00000370  03 08 00 05 00 00 00 53  74 6d 74 73 05 00 00 00  ··0·000S tmts·000
00000380  00 00 00 00 90 01 00 00  00 00 00 00 03 80 80 80  0000××00 0000·×××
00000390  50 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  P000·000 00000000
000003a0  00 00 00 00 01 00 00 00  00 00 00 00 06 00 00 00  0000·000 0000·000
000003b0  14 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  ·0000000 0000·000
000003c0  00 00 00 00 06 00 00 00  0e 00 00 00 06 00 00 00  0000·000 ·000·000
000003d0  17 00 00 00 74 00 00 00  01 00 00 00 2b 00 05 80  ·000t000 ·000+0·×
000003e0  80 80 80 80 50 00 00 00  02 00 00 00 00 00 00 00  ××××P000 ·0000000
```

1 Introduction

2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

```
➜  fictive-types-test clang-fictive-types-flow --data-dir="./tmp" --jobs 1

==================== Begin iteration       1 ====================
[T1/1] Working on lib.cpp
[T1/1] Working on main.cpp
               Error signalled when visiting.
               Error signalled when visiting.
[T1/1] No files remaining. Shutting down...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:16: error: use of undeclared built-in binary operator 'test_type' + 'test_type'
    int X2 = X + X + X;
             ~~^~~
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:14: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
             ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                         ~~~~^~~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:18: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                 ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                         ~~~~^~~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
Iteration begun Thu Apr 13 16:10:05 2023
Iteration ended Thu Apr 13 16:10:05 2023
Iteration duration 0 sec

====================  End  iteration       1 ====================

******************** At the beginning of iteration       2, no files are changed. ********************
➜  fictive-types-test
```

```
/local/repo/llvm-project/fictive-types-test/main.cpp:6:16: error: use of undeclared built-in binary operator 'test_type' + 'test_type'
    int X2 = X + X + X;
               ~~^~~
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:14: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
             ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:18: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                 ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
Iteration begun Thu Apr 13 16:14:08 2023
Iteration ended Thu Apr 13 16:14:08 2023
Iteration duration 0 sec

====================  End  iteration        1 ====================

******************** At the beginning of iteration        2, no files are changed. ********************
➜  fictive-types-test cat config.yaml
```

File: **config.yaml**

```
1  Operations:
2    Unary:
3    Binary:
4     - Operation: "test_type + test_type"
5       Result: "my_other_test_t"
```

```
➜  fictive-types-test
```

```
➜  fictive-types-test clang-fictive-types-flow --data-dir="./tmp" --jobs 1 --config-file "./config.yaml"

==================== Begin iteration        1 ====================
[T1/1] Working on lib.cpp
[T1/1] Working on main.cpp
               Error signalled when visiting.
[T1/1] No files remaining. Shutting down...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:20: error: use of undeclared built-in binary operator 'my_other_test_t' + 'test_type'
    int X2 = X + X + X;
             ~~~~~~~^~~
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:16: note: calculated as a result of built-in operator +
    int X2 = X + X + X;
             ~~^~~
<unknown>:0: remark: built-in operator + for 'test_type' and 'test_type' is defined to yield 'my_other_test_t'
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:14: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
             ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                         ~~~~^~~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:18: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
               ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                         ~~~~^~~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:22: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                 ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
```

```
/local/repo/llvm-project/fictive-types-test/main.cpp:6:18: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                 ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:22: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                     ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
Iteration begun Thu Apr 13 16:38:30 2023
Iteration ended Thu Apr 13 16:38:30 2023
Iteration duration 0 sec

====================  End  iteration        1 ====================

******************** At the beginning of iteration        2, no files are changed. ********************
➜  fictive-types-test cat config.yaml
```

File: **config.yaml**

```yaml
Operations:
  Unary:
  Binary:
   - Operation: "test_type + test_type"
     Result: "my_other_test_t"
   - Operation: "test_type + my_other_test_t"
     Result: "result_t"
     Commutative: true
```

```
➜  fictive-types-test
```

```
➜ fictive-types-test cat config.yaml
```

File: **config.yaml**

```yaml
Operations:
  Unary:
  Binary:
   - Operation: "test_type + test_type"
     Result: "my_other_test_t"
   - Operation: "test_type + my_other_test_t"
     Result: "result_t"
     Commutative: true
```

```
➜ fictive-types-test clang-fictive-types-flow --data-dir="./tmp" --jobs 1 --config-file "./config.yaml"

==================== Begin iteration        1 ====================
[T1/1] Working on lib.cpp
[T1/1] Working on main.cpp
[T1/1] No files remaining. Shutting down...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:9: warning: Var 'X2' declaration received Fictive Type 'result_t'
    int X2 = X + X + X;
    ~~~~^~~~~~~~~~~~~~
/local/repo/llvm-project/fictive-types-test/main.cpp:6:20: note: calculated as a result of built-in operator +
    int X2 = X + X + X;
             ~~~~~~~^~~
<unknown>:0: remark: built-in operator + for 'my_other_test_t' and 'test_type' is defined to yield 'result_t'
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:16: note: calculated as a result of built-in operator +
    int X2 = X + X + X;
             ~~^~~
<unknown>:0: remark: built-in operator + for 'test_type' and 'test_type' is defined to yield 'my_other_test_t'
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:14: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
             ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
```

```
/local/repo/llvm-project/fictive-types-test/main.cpp:6:16: note: calculated as a result of built-in operator +
    int X2 = X + X + X;
               ~~^~~
<unknown>:0: remark: built-in operator + for 'test_type' and 'test_type' is defined to yield 'my_other_test_t'
<unknown>:0: note: begin reasoning about the left-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:14: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
             ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:18: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                 ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
<unknown>:0: note: ... end reasoning about the left-hand side of the operator
<unknown>:0: note: begin reasoning about the right-hand side of the operator ...
/local/repo/llvm-project/fictive-types-test/main.cpp:6:22: note: carrying Fictive Type taint from referred declaration
    int X2 = X + X + X;
                     ^
/local/repo/llvm-project/fictive-types-test/main.cpp:5:46: remark: Var declaration here, tainted in the analysed source code
    [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
                                               ~~~~^~~~~~~~~~~~~~~~~~
<unknown>:0: note: ... end reasoning about the right-hand side of the operator
Iteration begun Thu Apr 13 16:39:21 2023
Iteration ended Thu Apr 13 16:39:21 2023
Iteration duration 0 sec

====================  End  iteration        1 ====================

******************** At the beginning of iteration        2, no files are changed. ********************
➜  fictive-types-test
```

1 Introduction

2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

The Timeline

- 2020 Feb.: Foundation stone laying ceremony, first commit for *Fictive Types* modelling in Clang
- **2021 Dec. 9**: C-level tool works as fix-point iteration, but does not scale
- Christmas & New Year 2021→2022: Implement custom data structure to make the tool scalable
- 2022 Mar.: Serialisation (`-collect`) prototype works

```
→ fictive-types-test cat main.cpp
```

File: **main.cpp**

```cpp
#include "header.h"

int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
{
    int X = lib::calc(ArgC);
    int X2 = X + X + X;

    return X2;
}
```

```
→ fictive-types-test
```

```
➜  fictive-types-test cat main.cpp
```

File: main.cpp

```
1  #include "header.h"
2
3  int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
4  {
5      int X = lib::calc(ArgC);
6      int X2 = X + X + X;
7
8      return X2;
9  }
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . ./main.cpp
In file included from main.cpp:1:
./header.h:7:16: warning: parameter should be attributed with fictive type 'test_type'
  int calc(int I);
               ^
           [[clang::fictive_type("test_type")]]
main.cpp:5:23: note: passing an expression that carries the fictive type as argument
    int X = lib::calc(ArgC);
                      ^~~~
main.cpp:3:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                                    ^
➜  fictive-types-test
```

```
➜  fictive-types-test cat main.cpp
```

File: main.cpp

```
1  #include "header.h"
2
3  int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
4  {
5      int X = lib::calc(ArgC);
6      int X2 = X + X + X;
7
8      return X2;
9  }
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . ./main.cpp
In file included from main.cpp:1:
./header.h:7:16: warning: parameter should be attributed with fictive type 'test_type'
  int calc(int I);
               ^
           [[clang::fictive_type("test_type")]]
main.cpp:5:23: note: passing an expression that carries the fictive type as argument
    int X = lib::calc(ArgC);
                      ^~~~
main.cpp:3:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                                    ^
➜  fictive-types-test clang-fictive-types-propagator -p . ./main.cpp
➜  fictive-types-test
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . ./main.cpp
In file included from main.cpp:1:
./header.h:7:16: warning: parameter should be attributed with fictive type 'test_type'
  int calc(int I);
               ^
           [[clang::fictive_type("test_type")]]
main.cpp:5:23: note: passing an expression that carries the fictive type as argument
    int X = lib::calc(ArgC);
                      ^~~~
main.cpp:3:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                                  ^
➜  fictive-types-test clang-fictive-types-propagator -p . ./main.cpp
➜  fictive-types-test clang-fictive-types-propagator -p . ./lib.cpp
lib.cpp:9:29: error: use of undeclared built-in binary operator '<no fictive type>' + 'test_type'
    DangerousGlobalVariable += I;
                            ^
lib.cpp:7:7: warning: function should be attributed with fictive type 'test_type'
  int calc(int I)
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:10:12: note: returning of an expression that carries the fictive type
    return I;
           ^
lib.cpp:7:16: note: expression references variable 'I' that has fictive type declared
  int calc(int I)
               ^
In file included from lib.cpp:1:
./header.h:7:7: warning: declaration related to previous diagnostic should be attributed with fictive type 'test_type'
  int calc([[clang::fictive_type("test_type")]] int I);
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:7:7: note: to keep in sync with the definition here
  int calc(int I)
      ^
➜  fictive-types-test
```

```
➜  fictive-types-test cat header.h
```

File: **header.h**

```
1  #pragma once
2
3  namespace lib
4  {
5    [[clang::fictive_type("global")]] extern int DangerousGlobalVariable;
6
7    int calc([[clang::fictive_type("test_type")]] int I);
8  }
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./lib.cpp
lib.cpp:7:7: warning: function should be attributed with fictive type 'test_type'
  int calc(int I)
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:10:12: note: returning of an expression that carries the fictive type
    return I;
           ^
lib.cpp:7:16: note: expression references variable 'I' that has fictive type declared
  int calc(int I)
               ^
In file included from lib.cpp:1:
./header.h:7:7: warning: declaration related to previous diagnostic should be attributed with fictive type 'test_type'
  int calc([[clang::fictive_type("test_type")]] int I);
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:7:7: note: to keep in sync with the definition here
  int calc(int I)
      ^
➜  fictive-types-test
```

```
➜  fictive-types-test cat header.h
```

File: **header.h**

```
1  #pragma once
2
3  namespace lib
4  {
5    [[clang::fictive_type("global")]] extern int DangerousGlobalVariable;
6
7    int calc([[clang::fictive_type("test_type")]] int I);
8  }
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./lib.cpp
lib.cpp:7:7: warning: function should be attributed with fictive type 'test_type'
  int calc(int I)
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:10:12: note: returning of an expression that carries the fictive type
    return I;
           ^
lib.cpp:7:16: note: expression references variable 'I' that has fictive type declared
  int calc(int I)
               ^
In file included from lib.cpp:1:
./header.h:7:7: warning: declaration related to previous diagnostic should be attributed with fictive type 'test_type'
  int calc([[clang::fictive_type("test_type")]] int I);
      ^
  [[clang::fictive_type("test_type")]]
lib.cpp:7:7: note: to keep in sync with the definition here
  int calc(int I)
      ^
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./lib.cpp
➜  fictive-types-test
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./main.cpp
main.cpp:5:9: warning: variable should be attributed with fictive type 'test_type'
    int X = lib::calc(ArgC);
        ^
    [[clang::fictive_type("test_type")]]
main.cpp:5:13: note: variable initialised from an expression that taints with fictive type
    int X = lib::calc(ArgC);
            ^~~~~~~~~~~~~~~
./header.h:7:44: note: the value of the called function 'calc' is declared to carry the fictive type
  [[clang::fictive_type("test_type")]] int calc([[clang::fictive_type("test_type")]] int I);
                                           ^
main.cpp:6:20: error: use of undeclared built-in binary operator '<no fictive type>' + 'test_type'
    int X2 = X + X + X;
                   ^
➜  fictive-types-test
```

1 Introduction

2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

```
→ fictive-types-test cat main.cpp
```

File: **main.cpp**

```cpp
namespace lib
{
  [[clang::fictive_type_inline]] int calc(int I)
  {
    return I;
  }
}

int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
{
    int X = lib::calc(ArgC);
    int X2 = X + X;

    return X2;
}
```

```
→ fictive-types-test
```

```
  7 | }
  8 |
  9 | int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
 10 | {
 11 |     int X = lib::calc(ArgC);
 12 |     int X2 = X + X;
 13 |
 14 |     return X2;
 15 | }
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./main.cpp
main.cpp:3:38: remark: summary created: function 'calc' returns the fictive type of the 1st parameter
  [[clang::fictive_type_inline]] int calc(int I)
                                     ^
main.cpp:5:12: note: returning of an expression that carries the fictive type
    return I;
           ^
main.cpp:3:47: note: expression references variable 'I' that is tainted in current execution
  [[clang::fictive_type_inline]] int calc(int I)
                                              ^
main.cpp:11:9: warning: variable should be attributed with fictive type 'test_type'
    int X = lib::calc(ArgC);
        ^
    [[clang::fictive_type("test_type")]]
main.cpp:11:13: note: variable initialised from an expression that taints with fictive type
    int X = lib::calc(ArgC);
            ^~~~~~~~~~~~~~~
main.cpp:11:13: remark: summary applied: the called function 'calc' has been calculated to return the fictive type of the 1st argument
    int X = lib::calc(ArgC);
            ^          ~~~~
main.cpp:9:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                              ^
main.cpp:12:16: error: use of undeclared built-in binary operator 'test_type' + 'test_type'
    int X2 = X + X;
               ^
➜  fictive-types-test
```

```
➜  fictive-types-test clang-fictive-types-propagator -p . --operation-map=./config.yaml ./main.cpp
main.cpp:3:38: remark: summary created: function 'calc' returns the fictive type of the 1st parameter
  [[clang::fictive_type_inline]] int calc(int I)
                                     ^
main.cpp:5:12: note: returning of an expression that carries the fictive type
    return I;
           ^
main.cpp:3:47: note: expression references variable 'I' that is tainted in current execution
  [[clang::fictive_type_inline]] int calc(int I)
                                              ^
main.cpp:11:9: warning: variable should be attributed with fictive type 'test_type'
    int X = lib::calc(ArgC);
        ^
    [[clang::fictive_type("test_type")]]
main.cpp:11:13: note: variable initialised from an expression that taints with fictive type
    int X = lib::calc(ArgC);
            ^~~~~~~~~~~~~~~
main.cpp:11:13: remark: summary applied: the called function 'calc' has been calculated to return the fictive type of the 1st argument
    int X = lib::calc(ArgC);
            ^          ~~~~
main.cpp:9:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                                  ^
main.cpp:12:16: error: use of undeclared built-in binary operator 'test_type' + 'test_type'
    int X2 = X + X;
               ^
➜  fictive-types-test cat config.yaml
```

File: **config.yaml**

```yaml
Unary:
Binary:
   - Operation: "test_type + test_type"
     Result: "test_type"
```

```
➜  fictive-types-test
```

```
    return I;
           ^
main.cpp:3:47: note: expression references variable 'I' that is tainted in current execution
  [[clang::fictive_type_inline]] int calc(int I)
                                              ^
main.cpp:9:5: warning: function should be attributed with fictive type 'test_type'
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
    ^
[[clang::fictive_type("test_type")]]
main.cpp:14:12: note: returning of an expression that carries the fictive type
    return X2;
           ^~
main.cpp:12:9: note: expression references variable 'X2' that is tainted in current execution
    int X2 = X + X;
        ^
main.cpp:11:9: warning: variable should be attributed with fictive type 'test_type'
    int X = lib::calc(ArgC);
        ^
    [[clang::fictive_type("test_type")]]
main.cpp:11:13: note: variable initialised from an expression that taints with fictive type
    int X = lib::calc(ArgC);
            ^~~~~~~~~~~~~~~
main.cpp:11:13: remark: summary applied: the called function 'calc' has been calculated to return the fictive type of the 1st argument
    int X = lib::calc(ArgC);
            ^         ~~~~
main.cpp:9:51: note: expression references variable 'ArgC' that has fictive type declared
int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
                                                  ^
main.cpp:12:9: warning: variable should be attributed with fictive type 'test_type'
    int X2 = X + X;
        ^
    [[clang::fictive_type("test_type")]]
main.cpp:12:16: note: variable initialised from an expression that taints with fictive type
    int X2 = X + X;
               ~~^~~
remark: built-in operator + call for 'test_type' and 'test_type' is defined to yield 'test_type'
➜  fictive-types-test
```

1 Introduction

2 Overview

3 In detail

4 Tooling & “Demo”
- Initial setup
- Propagation
- “Old” infrastructure
- Inlining
- Iterative driver

```
→ fictive-types-test cat main.cpp
```

File: main.cpp

```
 1  namespace lib
 2  {
 3    int calc(int I)
 4    {
 5      return I;
 6    }
 7  }
 8
 9  int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
10  {
11      int X = lib::calc(ArgC);
12      int X2 = X + X;
13
14      return X2;
15  }
```

```
→ fictive-types-test
```

```
11      int X = lib::calc(ArgC);
12      int X2 = X + X;
13
14      return X2;
15  }
```

```
➜  fictive-types-test clang-fictive-types ./compile_commands.json
Beginning next iteration...
[1/1] Running for main.cpp
/tmp/clang-fictive-types_-1c371e/compile_commands.json
/tmp/clang-fictive-types_-1c371e/main.cpp.-1.yaml
Performing rewrites...
Beginning next iteration...
[1/1] Running for main.cpp
/tmp/clang-fictive-types_-1c371e/compile_commands.json
No more replacements generated.
➜  fictive-types-test cat main.cpp
```

File: **main.cpp**

```
1   namespace lib
2   {
3     [[clang::fictive_type("test_type")]] int calc([[clang::fictive_type("test_type")]] int I)
4     {
5       return I;
6     }
7   }
8
9   int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
10  {
11      [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
12      int X2 = X + X;
13
14      return X2;
15  }
```

```
➜  fictive-types-test
```

```
→ fictive-types-test cat main.cpp
```

File: **main.cpp**

```
 1  namespace lib
 2  {
 3    [[clang::fictive_type_inline]] int calc(int I)
 4    {
 5      return I;
 6    }
 7  }
 8
 9  int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
10  {
11      int X = lib::calc(ArgC);
12      int X2 = X + X;
13
14      return X2;
15  }
```

```
→ fictive-types-test
```

```
 11      int X = lib::calc(ArgC);
 12      int X2 = X + X;
 13
 14      return X2;
 15  }
```

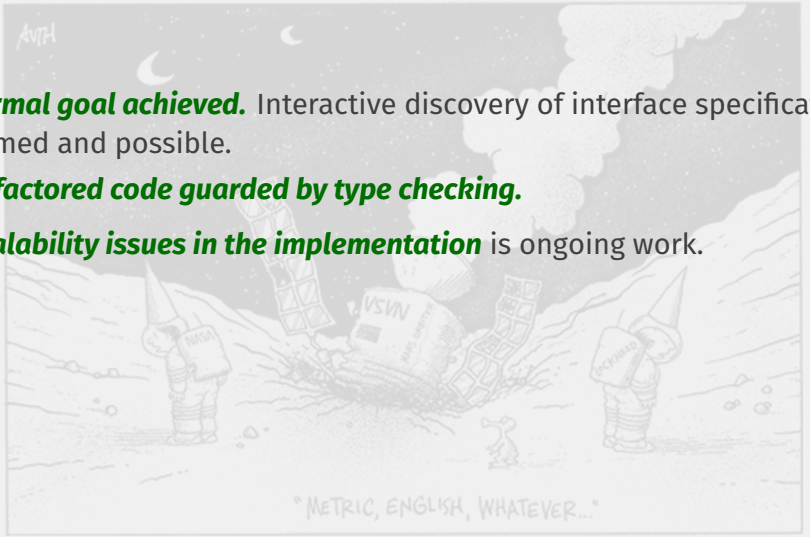
```
➜  fictive-types-test clang-fictive-types ./compile_commands.json
Beginning next iteration...
[1/1] Running for main.cpp
/tmp/clang-fictive-types_-4a4f25/compile_commands.json
/tmp/clang-fictive-types_-4a4f25/main.cpp.-1.yaml
Performing rewrites...
Beginning next iteration...
[1/1] Running for main.cpp
/tmp/clang-fictive-types_-4a4f25/compile_commands.json
No more replacements generated.
➜  fictive-types-test cat main.cpp
```

File: **main.cpp**

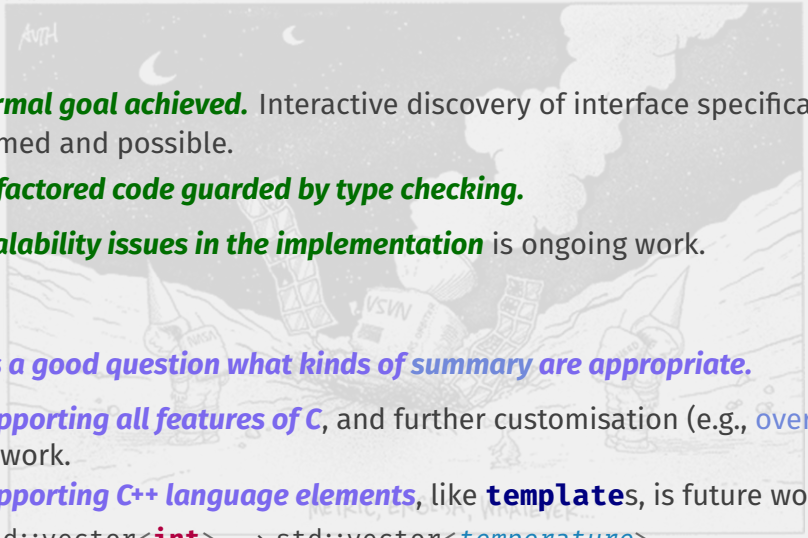
```
  1  namespace lib
  2  {
  3    [[clang::fictive_type_inline]] int calc(int I)
  4    {
  5      return I;
  6    }
  7  }
  8
  9  int main([[clang::fictive_type("test_type")]] int ArgC, char* ArgV[])
 10  {
 11      [[clang::fictive_type("test_type")]] int X = lib::calc(ArgC);
 12      int X2 = X + X;
 13
 14      return X2;
 15  }
```

```
➜  fictive-types-test
```

- ✓ ***Formal goal achieved.*** Interactive discovery of interface specification is performed and possible.
- ✓ ***Refactored code guarded by type checking.***
- ✓ ***Scalability issues in the implementation*** is ongoing work.



Remember the Mars Climate Orbiter incident from 1999?



- ✓ ***Formal goal achieved.*** Interactive discovery of interface specification is performed and possible.
- ✓ ***Refactored code guarded by type checking.***
- ✓ ***Scalability issues in the implementation*** is ongoing work.

- ***It's a good question what kinds of summary are appropriate.***
- ***Supporting all features of C***, and further customisation (e.g., overloads) is future work.
- ***Supporting C++ language elements***, like **`template`**s, is future work.
  - `std::vector<int>` ⟶ `std::vector<temperature>`

Remember the Mars Climate Orbiter incident from 1999?