Higher-Order Template Metaprogramming

Higher-Order Template Metaprogramming (in C++23)

Higher-Order Template Metaprogramming (in C++23)

Don’t do this… yet.

	Logic		
0th order	P		
1st order	$P(x)$		
2nd order	$\exists\, P\; P(b)$		

	Logic	FP	
0th order	P	x	
1st order	P(x)	\x. x + 1	
2nd order	∃ P P(b)	\fx. f(f x)	

	Logic	FP	TMP
0th order	P	x	<int I>
1st order	P(x)	\x. x + 1	<class C>
2nd order	∃ P P(b)	\fx. f(f x)	?

	Logic	FP	TMP
0th order	P	x	`<int I>`
1st order	P(x)	\x. x + 1	`<class C>`
2nd order	∃ P P(b)	\fx. f(f x)	`<template<class> class>`

	Logic	FP	TMP
0th order	P	x	`<int I>`
1st order	P(x)	\x. x + 1	`<class C>`

2nd order	∃ P P(b)	\fx. f(f x)	`<template<class> class>`
			`<template<class> bool>`

	Logic	FP	TMP
0th order	P	x	`<int I>`
1st order	P(x)	\x. x + 1	`<class C>`

2nd order	∃ P P(b)	\fx. f(f x)	`<template<class> class>`
			`<template<class> bool>`
			`<template<class> concept>`

	Logic	FP	TMP
0th order	P	x	<int I>
1st order	P(x)	\x. x + 1	<class C>
2nd order	∃ P P(b)	\fx. f(f x)	<template<class> class> <template<class> bool> <template<class> concept>

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

“Naming things is hard.”

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

BANNED

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

BANNED

template

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

BANNED

template

BANNED

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

BANNED

template

BANNED

decltype

```
enum class E { a, b, c, };
std::is_enum_v auto e = E::a;
auto f(std::is_enum_v auto e) {}
std::is_enum_v auto f() { return E::a; }
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum_v;
};
```

static_assert

BANNED

template

BANNED

decltype

BANNED

```
enum class E { a, b, c, };

// ...

template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> std::is_enum;
};
```

```cpp
template<class T, template<class> class TT>
concept Trait = TT<T>::value;

template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
};
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
};
```

```cpp
struct S {
    E code();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> ???;
};
```

```
struct S {
    E code();
    void run();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
};
```

```
struct S {
    E code();
    void run();
};
A auto s = S();
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> std::same_as<void>;
};
```

```cpp
struct S {
    E code();
    void run();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> std::same_as<void>;
};
```

```
struct S {
    E code();
    void const run();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
};
```

```
struct S {
    E code();
    void const run();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> ???;
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
};
```

```cpp
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
};

template<class T, class U>
concept RangeOf = std::ranges::range<T> and
    std::same_as<std::ranges::range_value_t<T>, U>;
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<std::integral>;
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<std::integral>;
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
};
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
};
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
};

template<class T, auto C>
concept RangeOver = std::ranges::range<T> and requires {
    C.template operator()<std::ranges::range_value_t<T>>();
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
};

template<class T, auto C>
concept Satisfies = requires {
    C.template operator()<T>();
};

template<class T, auto C>
concept RangeOver = std::ranges::range<T> and
    Satisfies<std::ranges::range_value_t<T>, C>;
```

```cpp
struct S {
    E code();
    void const run();
    std::span<int> ids();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> ???;
 };
```

```
struct B {};
struct D : B {};
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D d() const;
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> std::derived_from<B>;
 };
```

```
struct B {};
struct D : B {};
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D d() const;
};
A auto s = S();
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> std::derived_from<B>;
 };
```

```cpp
struct B {};
struct D : B {};
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D const& d() const;
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> SatisfiesAfter<std::remove_cvref,
        []<std::derived_from<B>>{}>;
};

template<class T, template<class> class TT, auto C>
concept SatisfiesAfter =
    Satisfies<typename TT<T>::type, C>;
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D const& d() const;
};
A auto s = S();
```

```cpp
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> SatisfiesAfter<std::remove_cvref,
        []<std::derived_from<B>>{}>;
    { tc.d() } -> Satisfies<[]<class U> requires
        std::derived_from<std::remove_cvref_t<U>, B> {}>;
};
```

```cpp
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D const& d() const;
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> SatisfiesAfter<std::remove_cvref,
        []<std::derived_from<B>>{}>;
    { tc.d() } -> Satisfies<[]<class U> requires
        std::derived_from<std::remove_cvref_t<U>, B> {}>;
    { t.elapsed() } -> ???;
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D const& d() const;
    std::chrono::picoseconds
        elapsed();
};
A auto s = S();
```

```
template<class T>
concept A = requires(T t, T const tc) {
    { t.code() } -> Trait<std::is_enum>;
    { t.run() } -> Trait<std::is_void>;
    { t.ids() } -> RangeOf<int>;
    { t.ids() } -> RangeOver<[]<std::integral>{}>;
    { tc.d() } -> SatisfiesAfter<std::remove_cvref,
        []<std::derived_from<B>>{}>;
    { tc.d() } -> Satisfies<[]<class U> requires
        std::derived_from<std::remove_cvref_t<U>, B> {}>;
    { t.elapsed() } -> Satisfies<[]<class U> requires
        requires(U u) { []<std::integral R, auto D>(
            std::chrono::duration<R, std::ratio<1, D>>){}(
                u);
        } {}>;
};
```

```
struct S {
    E code();
    void const run();
    std::span<int> ids();
    D const& d() const;
    std::chrono::picoseconds
        elapsed();
};
A auto s = S();
```

```
<source>:33:30: internal compiler error: Segmentation fault
  33 |     { t.ids() } -> RangeOver<[]<std::integral>{}>;
     |                              ^~~~~~~~~~~~~~~~~~~
0x255cc4e internal_error(char const*, ...)
    ???:0
0xceb951 template_parms_to_args(tree_node*)
    ???:0
0xd29d62 tsubst_lambda_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xd0e051 tsubst_template_arg(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*, tree_node*,
int, auto_deduction_context, tree_node*, int, tree_node*)
    ???:0
0xb701d6 tsubst_requires_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*, tree_node*,
int, auto_deduction_context, tree_node*, int, tree_node*)
    ???:0
0xbd4e9d cp_finish_decl(tree_node*, tree_node*, bool, tree_node*,
int, cp_decomp*)
    ???:0
0xcdc754 c_parse_file()
    ???:0
0xe1eca9 c_common_parse_file()
    ???:0
Please submit a full bug report, with preprocessed source (by
using -freport-bug).
Please include the complete backtrace with any bug report.
See <https://gcc.gnu.org/bugs/> for instructions.
Compiler returned: 1
```

```
<source>:33:30: internal compiler error: Segmentation
fault
   33 |       { t.ids() } -> RangeOver<[]<std::integral>{}>;
      |                                    ^~~~~~~~~~~~~~~~~~~
0x255cc4e internal_error(char const*, ...)
    ???:0
0xceb951 template_parms_to_args(tree_node*)
    ???:0
0xd29d62 tsubst_lambda_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xd0e051 tsubst_template_arg(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*,
tree_node*, int, auto_deduction_context, tree_node*, int,
tree_node*)
    ???:0
0xb701d6 tsubst_requires_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*,
tree_node*, int, auto_deduction_context, tree_node*, int,
tree_node*)
    ???:0
0xbd4e9d cp_finish_decl(tree_node*, tree_node*, bool,
tree_node*, int, cp_decomp*)
    ???:0
0xcdc754 c_parse_file()
    ???:0
0xe1eca9 c_common_parse_file()
    ???:0
Please submit a full bug report, with preprocessed source
(by using -freport-bug).
Please include the complete backtrace with any bug report.
See <https://gcc.gnu.org/bugs/> for instructions.
Compiler returned: 1
```

```
clang++:
/root/llvm-project/clang/lib/AST/ExprConstant.cpp:15495:
bool
clang::Expr::EvaluateAsConstantExpr(clang::Expr::EvalResul
t&, const clang::ASTContext&,
clang::Expr::ConstantExprKind) const: Assertion
`!isValueDependent() && "Expression evaluator can't be
called on a dependent expression."' failed.
PLEASE submit a bug report to
https://github.com/llvm/llvm-project/issues/ and include
the crash backtrace, preprocessed source, and associated
run script.
Stack dump:
0.  Program arguments:
/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
-gdwarf-4 -g -o /app/output.s -mllvm
--x86-asm-syntax=intel -S
--gcc-toolchain=/opt/compiler-explorer/gcc-snapshot
-fcolor-diagnostics -fno-crash-diagnostics -std=c++2b -O
-stdlib=libc++ <source>
1.  <source>:47:15: current parser token ';'
#0 0x0000000003722508
llvm::sys::PrintStackTrace(llvm::raw_ostream&, int)
(/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
+0x3722508)
#1 0x00000000037201cc llvm::sys::CleanupOnSignal(unsigned
long)
(/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
+0x37201cc)
#2 0x0000000003668e38 CrashRecoverySignalHandler(int)
CrashRecoveryContext.cpp:0:0
#3 0x00007fd43e6ce420 __restore_rt
(/lib/x86_64-linux-gnu/libpthread.so.0+0x14420)
#4 0x00007fd43e19100b raise
(/lib/x86_64-linux-gnu/libc.so.6+0x4300b)
#5 0x00007fd43e170859 abort
(/lib/x86_64-linux-gnu/libc.so.6+0x22859)
#6 0x00007fd43e170729
(/lib/x86_64-linux-gnu/libc.so.6+0x22729)
#7 0x00007fd43e181fd6
(/lib/x86_64-linux-gnu/libc.so.6+0x33fd6)
```

```
<source>:33:30: internal compiler error: Segmentation
fault
  33 |     { t.ids() } -> RangeOver<[]<std::integral>{}>;
     |                                   ^~~~~~~~~~~~~~~~~~
0x255cc4e internal_error(char const*, ...)
    ???:0
0xceb951 template_parms_to_args(tree_node*)
    ???:0
0xd29d62 tsubst_lambda_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xd0e051 tsubst_template_arg(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*,
tree_node*, int, auto_deduction_context, tree_node*, int,
tree_node*)
    ???:0
0xb701d6 tsubst_requires_expr(tree_node*, tree_node*, int,
tree_node*)
    ???:0
0xb71b22 constraints_satisfied_p(tree_node*, tree_node*)
    ???:0
0xcf8c9b do_auto_deduction(tree_node*, tree_node*,
tree_node*, int, auto_deduction_context, tree_node*, int,
tree_node*)
    ???:0
0xbd4e9d cp_finish_decl(tree_node*, tree_node*, bool,
tree_node*, int, cp_decomp*)
    ???:0
0xcdc754 c_parse_file()
    ???:0
0xe1eca9 c_common_parse_file()
    ???:0
Please submit a full bug report, with preprocessed source
(by using -freport-bug).
Please include the complete backtrace with any bug report.
See <https://gcc.gnu.org/bugs/> for instructions.
Compiler returned: 1
```

```
clang++:
/root/llvm-project/clang/lib/AST/ExprConstant.cpp:15495:
bool
clang::Expr::EvaluateAsConstantExpr(clang::Expr::EvalResul
t&, const clang::ASTContext&,
clang::Expr::ConstantExprKind) const: Assertion
`!isValueDependent() && "Expression evaluator can't be
called on a dependent expression."' failed.
PLEASE submit a bug report to
https://github.com/llvm/llvm-project/issues/ and include
the crash backtrace, preprocessed source, and associated
run script.
Stack dump:
0.  Program arguments:
/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
-gdwarf-4 -g -o /app/output.s -mllvm
--x86-asm-syntax=intel -S
--gcc-toolchain=/opt/compiler-explorer/gcc-snapshot
-fcolor-diagnostics -fno-crash-diagnostics -std=c++2b -O
-stdlib=libc++ <source>
1.  <source>:47:15: current parser token ';'
#0 0x0000000003722508
llvm::sys::PrintStackTrace(llvm::raw_ostream&, int)
(/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
+0x3722508)
#1 0x00000000037201cc llvm::sys::CleanupOnSignal(unsigned
long)
(/opt/compiler-explorer/clang-assertions-trunk/bin/clang++
+0x37201cc)
#2 0x0000000003668e38 CrashRecoverySignalHandler(int)
CrashRecoveryContext.cpp:0:0
#3 0x00007fd43e6ce420 __restore_rt
(/lib/x86_64-linux-gnu/libpthread.so.0+0x14420)
#4 0x00007fd43e19100b raise
(/lib/x86_64-linux-gnu/libc.so.6+0x4300b)
#5 0x00007fd43e170859 abort
(/lib/x86_64-linux-gnu/libc.so.6+0x22859)
#6 0x00007fd43e170729
(/lib/x86_64-linux-gnu/libc.so.6+0x22729)
#7 0x00007fd43e181fd6
(/lib/x86_64-linux-gnu/libc.so.6+0x33fd6)
```

```
example.cpp
Compiler returned: 0
```