Concepts Error Messages for Humans

Document: ISO/IEC JTC1/SC22/WG21 P2429R0
Date: 2022-05-16
Audience: SG15 (Tooling)
Reply to: Sy Brand (sybrand@microsoft.com)

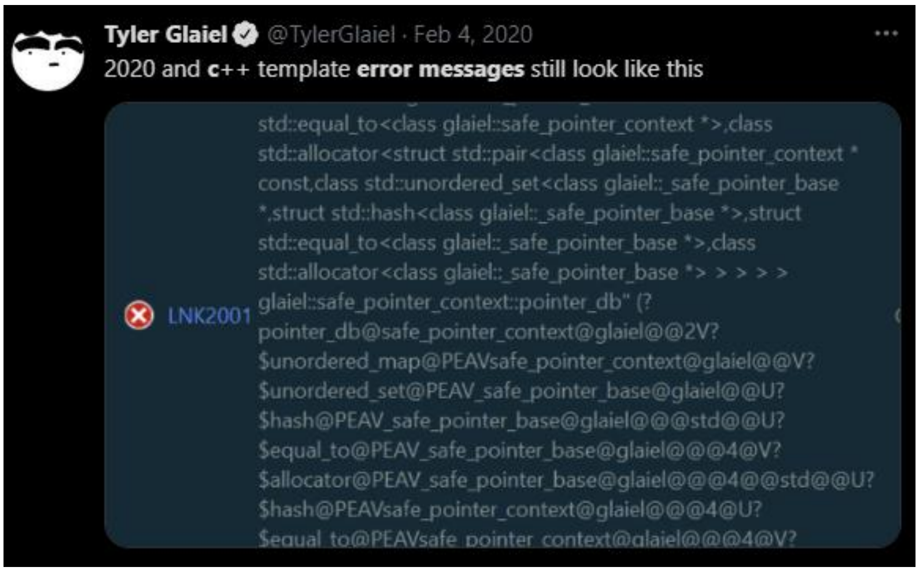
Abstract

This paper aims to prompt discussion about what is desirable and feasible for improving the diagnostics output by C++ tools. It:

- summarizes the state of the art in compiler error message research,
- gives examples of what production compilers are doing to improve their error messages,
- presents the current state of error messages for concepts-based C++ programming in MSVC, GCC, and Clang, and
- suggests potential improvements to the state of the art.

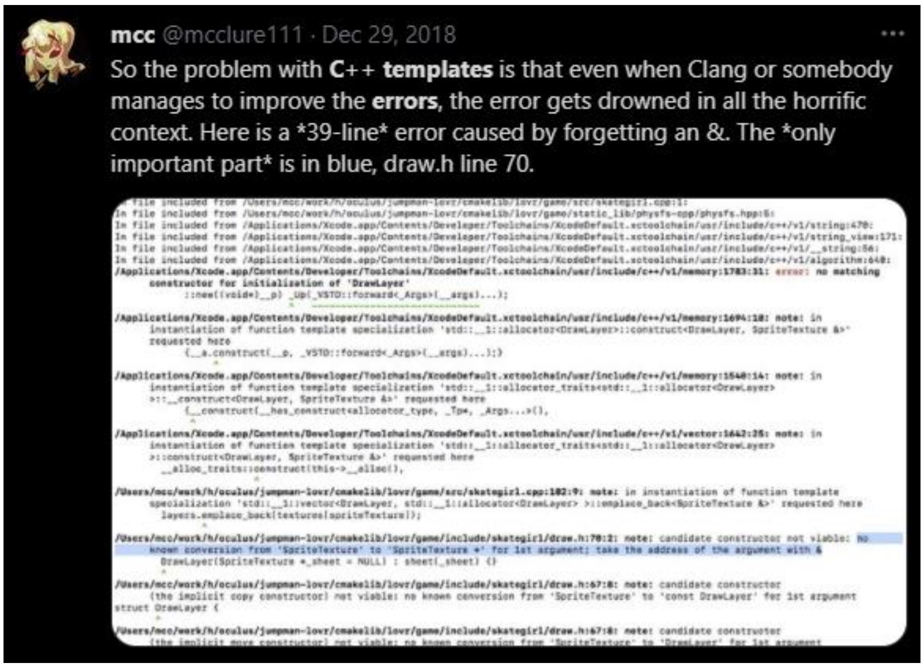
Motivation

C++ addict @janwilmans · Feb 29, 2020
c++ compiler errors be like



Tyler Glaiel @TylerGlaiel · Feb 4, 2020
2020 and c++ template error messages still look like this

```
LNK2001 std::equal_to<class glaiel::safe_pointer_context *>,class
        std::allocator<struct std::pair<class glaiel::safe_pointer_context *
        const,class std::unordered_set<class glaiel::_safe_pointer_base
        *,struct std::hash<class glaiel::_safe_pointer_base *>,struct
        std::equal_to<class glaiel::_safe_pointer_base *>,class
        std::allocator<class glaiel::_safe_pointer_base *> > > > >
        glaiel::safe_pointer_context::pointer_db" (?
        pointer_db@safe_pointer_context@glaiel@@2V?
        $unordered_map@PEAVsafe_pointer_context@glaiel@@V?
        $unordered_set@PEAV_safe_pointer_base@glaiel@@U?
        $hash@PEAV_safe_pointer_base@glaiel@@@std@@U?
        $equal_to@PEAV_safe_pointer_base@glaiel@@@4@V?
        $allocator@PEAV_safe_pointer_base@glaiel@@@4@@std@@U?
        $hash@PEAVsafe_pointer_context@glaiel@@@4@U?
        $equal_to@PEAVsafe_pointer_context@glaiel@@@4@V?
```



mcc @mcclure111 · Dec 29, 2018

So the problem with **C++ templates** is that even when Clang or somebody manages to improve the **errors**, the error gets drowned in all the horrific context. Here is a *39-line* error caused by forgetting an &. The *only important part* is in blue, draw.h line 70.

```
In file included from /Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/src/skategirl.cpp:1:
In file included from /Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/static_lib/physfs-cpp/physfs.hpp:6:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/string:470:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/string_view:171:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/__string:56:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/algorithm:640:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/memory:1783:31: error: no matching
      constructor for initialization of 'DrawLayer'
            ::new((void*)__p) _Up(_VSTD::forward<_Args>(__args)...);
                              ^   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/memory:1694:18: note: in
      instantiation of function template specialization 'std::__1::allocator<DrawLayer>::construct<DrawLayer, SpriteTexture &>'
      requested here
            {__a.construct(__p, _VSTD::forward<_Args>(__args)...);}
                 ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/memory:1540:14: note: in
      instantiation of function template specialization 'std::__1::allocator_traits<std::__1::allocator<DrawLayer>
      >::__construct<DrawLayer, SpriteTexture &>' requested here
            {__construct(__has_construct<allocator_type, _Tp*, _Args...>(),
             ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1/vector:1642:25: note: in
      instantiation of function template specialization 'std::__1::allocator_traits<std::__1::allocator<DrawLayer>
      >::construct<DrawLayer, SpriteTexture &>' requested here
        __alloc_traits::construct(this->__alloc(),
                        ^
/Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/src/skategirl.cpp:182:9: note: in instantiation of function template
      specialization 'std::__1::vector<DrawLayer, std::__1::allocator<DrawLayer> >::emplace_back<SpriteTexture &>' requested here
        layers.emplace_back(textures[spriteTexture]);
               ^
/Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/include/skategirl/draw.h:70:2: note: candidate constructor not viable: no
      known conversion from 'SpriteTexture' to 'SpriteTexture *' for 1st argument; take the address of the argument with &
        DrawLayer(SpriteTexture *_sheet = NULL) : sheet(_sheet) {}
        ^
/Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/include/skategirl/draw.h:67:8: note: candidate constructor
      (the implicit copy constructor) not viable: no known conversion from 'SpriteTexture' to 'const DrawLayer' for 1st argument
struct DrawLayer {
       ^
/Users/mcc/work/h/oculus/jumpman-lovr/cmakelib/lovr/game/include/skategirl/draw.h:67:8: note: candidate constructor
      (the implicit move constructor) not viable: no known conversion from 'SpriteTexture' to 'DrawLayer' for 1st argument
```

In every ISOCPP survey so far, compiler errors have been one of the top answers for "what is one thing you would change about C++ if you had a magic wand".

A 2017 study[1] found:

- Programmers do read error messages;
- the difficulty of reading these messages is comparable to the difficulty of reading source code;
- difficulty reading error messages significantly predicts task performance, and;
- participants allocate a substantial portion of their total task to reading error messages (13-25%).

In summary: the quality of error messages is an important factor of a compiler's job, and C++ compilers have a lot of room for improvement. Concepts give us an opportunity to improve.

Aside: compiler vendors have put a lot of time, thought, and effort into improving their diagnostics. This work is greatly appreciated, and this paper does not intend to disparage any of it, only to improve the state of the art further.

Guidelines

The paper "Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research"[2] carried out a survey of the state-of-the-art of compiler error message research and put together a set of guidelines based on this survey:

- **Increase readability** by using plain language, being concise, and writing errors for humans rather than tools.
- **Reduce cognitive load** by placing relevant information near the offending code, reducing redundancy so the user does not process the same information twice, and using multiple modalities to provide feedback.
- **Provide context** that can help the user.
- **Use a positive tone**
    - "when novices encounter violent messages ..., vague phrases ... , or obscure codes ... , they are understandably shaken, confused, dismayed, and discouraged from continuing"
- **Show examples** of similar errors that are minimal and aid understanding.

[1] Do Developers Read Compiler Error Messages?
[2] Compiler Error Messages Considered Unhelpful

- **Show solutions or hints.**
- **Allow dynamic interaction** by providing the user with autonomy over error message presentation.
- **Provide scaffolding** that helps the user connect concepts in the language with errors in their code.
- **Use logical argumentation** by providing a coherent narrative of the error.
- **Report errors at the right time** by giving the user the right amount of information when they need it.

I'll build out these descriptions a bit in how they can be applied to C++ and what other languages are doing.

Readability

Most resources on compiler error messages state "readability" as a key factor of message design, but there's very little guidance or studies on what that means in practice.

A 2021 empirical study[3] on error readability came to the following conclusions:

- Experts, non-experts and students assess the readability of messages differently.
- Error messages from different languages are perceived to have different levels of readability.
- Shorter messages tend to be more readable.
- Messages phrased positively tend to be more readable.
- Messages with more jargon and acronyms tend to be less readable.

Cognitive Load

The most extensive work in this area[4] provides three guidelines to reduce cognitive load in programmers receiving error messages to maximise working memory: place relevant information near the offending code, reduce redundancy so the user does not process the same information twice, and use multiple modalities to provide feedback.

To minimise cognitive load for C++ errors, IDEs/text editors often provide in-editor “squiggles”, error summarization windows, and console output as multiple communication modalities. The squiggles also help place the relevant information near the offending code.

Context

From "What the Compiler Should Tell the User"[5] (1974!), which is an oft-cited text in compiler error literature.

> One of the hardest things to remember in designing error diagnostics is that you don't know what the error was. Two (or more) pieces of information have been found to be inconsistent, but it cannot be said with certainty where the error lies. The safest strategy is to describe the symptom (the detected inconsistency) as clearly as possible before attempting to make any suggestions about the nature of the error.

For concepts-related diagnostics, I think the most important pieces of context that can be given to the user are:

1. Which constraints failed and why.
2. If there were multiple candidates considered, what were they and why were they discarded.

---

[3] Towards Assessing the Readability of Programming Error Messages
[4] IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda
[5] What the Compiler Should Tell the User

Positive tone

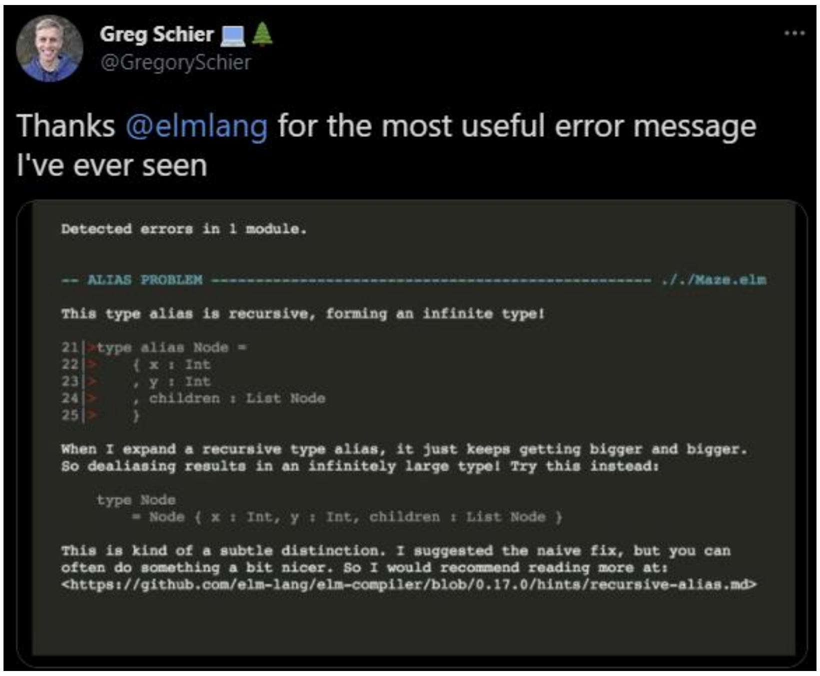
The delightfully-titled paper "How a computer should talk to people"[6] has this to say:

> We learn about and use a computer more effectively when we feel secure and experience success with it. What helps us feel at ease with a computer program or system?
>
> Friendliness helps people feel at ease. And it is not difficult to be "friendly". It does not require special gestures or mannerisms. Just provide a helpful message – one that lets people know what is happening now so they can predict what will happen next, or one that lets people actually control what will happen next by their response to the message.
>
> The main way we are unfriendly to other people is to ignore them. Another way we are unfriendly is to give an obscure message, because it can threaten people who are already insecure. They may think they are incapable of understanding (when, in fact, the message cannot be understood).
>
> Various negative tones or actions are unfriendly: being manipulative, not giving a second chance, talking down, using fashionable slang, blaming. We must not seem to blame the person. We should avoid suggesting that the person is inadequate. Phrases like "you forgot" may seem harmless, but what if a computer said this to you four or five times in two minutes? Anyway, the person may disagree, so why risk offense?

Elm[7] attempts very strongly to tackle this. Its tagline is "A *delightful* language for reliable web applications" (emphasis mine). A blog post on their error messages[8] states:

> Compilers should be assistants, not adversaries

Its error messages are detailed, conversational, and written in the first person:



Greg Schier
@GregorySchier

Thanks @elmlang for the most useful error message I've ever seen

```
Detected errors in 1 module.


-- ALIAS PROBLEM -------------------------------------------------- ././Maze.elm

This type alias is recursive, forming an infinite type!

21|>type alias Node =
22|>    { x : Int
23|>    , y : Int
24|>    , children : List Node
25|>    }

When I expand a recursive type alias, it just keeps getting bigger and bigger.
So dealiasing results in an infinitely large type! Try this instead:

    type Node
        = Node { x : Int, y : Int, children : List Node }

This is kind of a subtle distinction. I suggested the naive fix, but you can
often do something a bit nicer. So I would recommend reading more at:
<https://github.com/elm-lang/elm-compiler/blob/0.17.0/hints/recursive-alias.md>
```

[6] How a computer should talk to people
[7] Elm lang website
[8] Compilers as assistants

Examples

In the above Elm error you can see how they used an example to show potential solutions. This is considerably more difficult for C++.

For specific concept failures it may be possible to give examples. E.g. if `std::bidirectional_range` fails, the diagnostic could include examples of the operations which must be supported, and could use `std::list` vs `std::forward_list` as an illustrative example.

Solutions and Hints

Again, giving solutions and hints for C++ is considerably more difficult than in other languages.

GCC gives examples for some naming typos, e.g.:

```cpp
struct person {
    int age;
};

int main() {
    person sy;
    sy.agee;
}
```

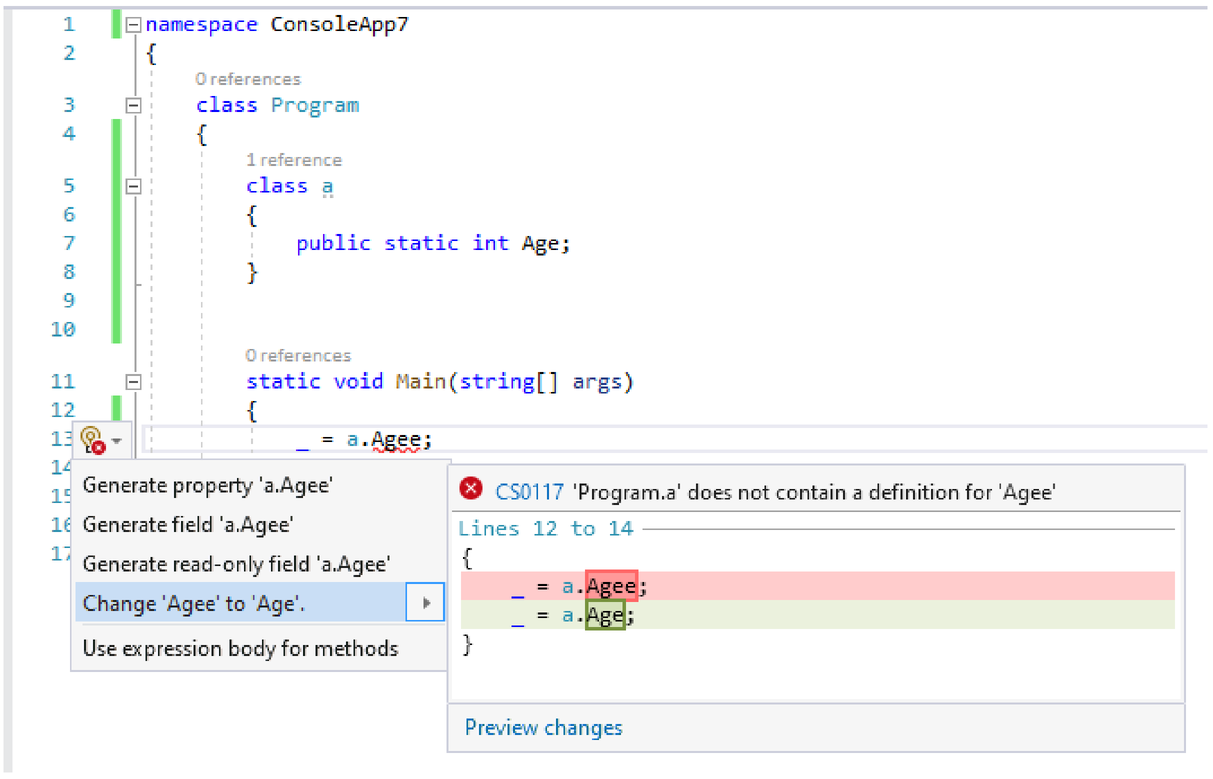
```
<source>: In function 'int main()':
<source>:7:8: error: 'struct person' has no member named 'agee'; did you mean
'age'?
    7 |     sy.agee;
      |        ^~~~
      |        age
```

GCC also can suggest `#include`s for standard library headers, similar to how we do for quick fixes:

```
<source>:7:14: error: 'cout' is not a member of 'std'
    7 |         std::cout << i;
      |              ^~~~
<source>:2:1: note: 'std::cout' is defined in header '<iostream>'; did you
forget to '#include <iostream>'?
    1 | #include <vector>
  +++ |+#include <iostream>
    2 |
```

Again, for specific concepts failures some hints could be given. Like if `std::borrowed_range` fails, the diagnostic could suggest ensuring that you are not operating on a function call expression returning by value, like `get_vector() | std::views::drop(2);`.

C# handles the 'spell check' scenario in the IDE by providing suggestions in the IDE:


```
namespace ConsoleApp7
{
    0 references
    class Program
    {
        1 reference
        class a
        {
            public static int Age;
        }


        0 references
        static void Main(string[] args)
        {
            _ = a.Agee;
```

Generate property 'a.Agee'
Generate field 'a.Agee'
Generate read-only field 'a.Agee'
Change 'Agee' to 'Age'.
Use expression body for methods

CS0117 'Program.a' does not contain a definition for 'Agee'

```
Lines 12 to 14
{
    _ = a.Agee;
    _ = a.Age;
}
```

Dynamic Interaction

Users can interact dynamically with the errors by expanding and contracting descriptions in the error list, or by passing flags to change warning levels.

GCC has the `-fconcepts-diagnostic-depth` flag which allows users to control how deep the diagnostics of concepts failure diagnostics will go into the instantiation stack.

Scaffolding

The kind of scaffolding referred to here is logical scaffolding where the user builds up a mental picture of how their code relates to language concepts, such that they can understand their error and deepen their comprehension of the language. This is where there's a tension between using jargon terms and simple language: the simple language can make the diagnostic more readable, but the jargon terms may aid in building mental scaffolding if it's clear how they relate to the code.

As such, when using jargon terms in errors, we must be especially careful to make it clear what they're referring to in the code.

Logical Argumentation

Logical argumentation maps onto concepts error messages through constraint failure stacks. I.e. the messages provide logical argumentation by saying "Constraint A failed because it consists of the disjunction of Constraints B and C; Constraint B failed because […], Constraint C failed because […]". Providing this context allows users to build up a mental model of what caused the failure.

Timing

IDEs often have two main times at which they diagnose errors:

- During design-time in the editor, through in-editor squiggles
- When a build fails, through an error window

Ideally, concepts failures can be diagnosed at the earliest moment possible using the feedback options available to tool vendors.

Other Languages

Elm

Elm has put a *lot* of effort and thought into their error messages. They've produced two blog posts about improvements they made, including the design behind them[9][10].

Take this program, which attempts to display the text 4 to the user.

```
main = text 4
```

Compiling this code results in the following error.

```
The 1st argument to `text` is not what I expect:
5|   text 4
          ^
This argument is a number of type:
    number
But `text` needs the 1st argument to be:
    String
Hint: Try using String.fromInt to convert it to a string?
```

Comparing this to the list of guidelines above, it fulfils a remarkable number of them. The error is clear and readable, giving relevant context, and a potential solution, all delivered with a positive tone.

Consider this slightly more complex program:

```
sy =
  { first = "Sy"
  , last = "Brand"
  }

isOver50 person =
  person.age > 50

answer = isOver50 sy
```

Compiling the above code gives this error:

```
The 1st argument to `isOver50` is not what I expect:

9| answer = isOver50 sy
                     ^^
This `sy` value is a:

    { first : String, last : String }

But `isOver50` needs the 1st argument to be:

    { a | age : number }

Hint: Seems like a record field typo. Maybe age should be last?
```

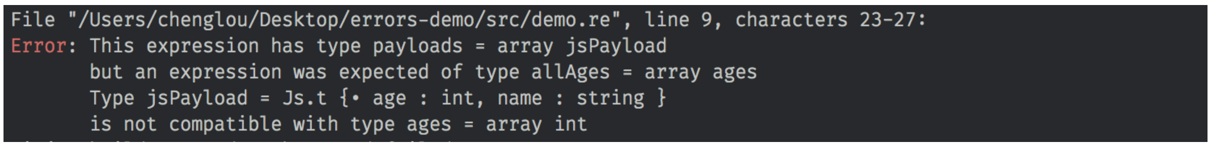
[9] Error messages for humans
[10] Compilers as assistants

```
Hint: Can more type annotations be added? Type annotations always help me
give
more specific messages, and I think they could help a lot in this case!
```

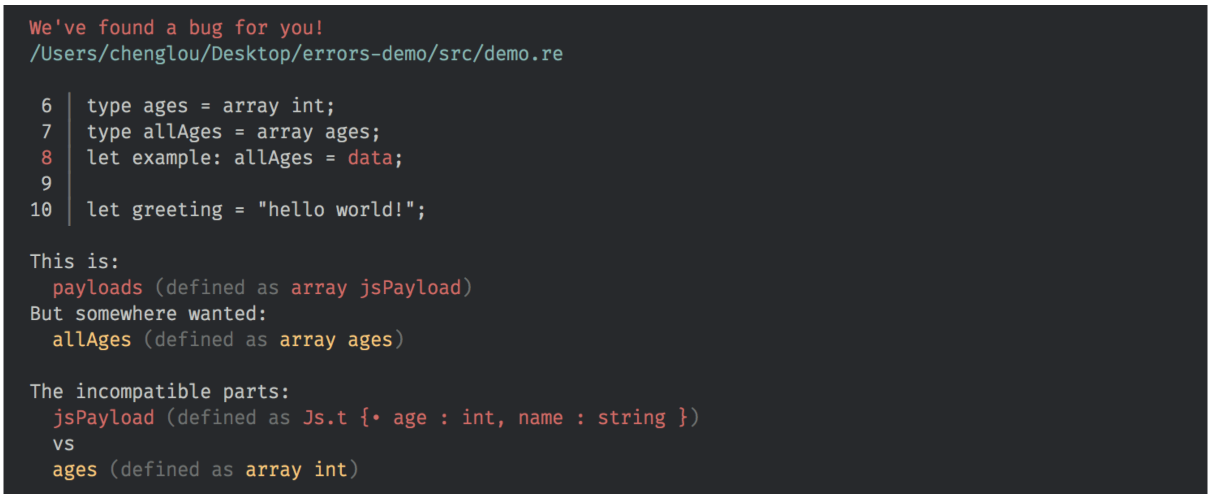
The hints are not *quite* right, in that it's a missing field in my type rather than a typo in the usage, but in general this error message is a breath of fresh air.

ReasonML

Here's before-and-after pictures of a huge error message overhaul which ReasonML carried out in 2017, as written about in their blog post[11]:



```
File "/Users/chenglou/Desktop/errors-demo/src/demo.re", line 9, characters 23-27:
Error: This expression has type payloads = array jsPayload
       but an expression was expected of type allAges = array ages
       Type jsPayload = Js.t {• age : int, name : string }
       is not compatible with type ages = array int
```



```
We've found a bug for you!
/Users/chenglou/Desktop/errors-demo/src/demo.re

  6 | type ages = array int;
  7 | type allAges = array ages;
  8 | let example: allAges = data;
  9 |
 10 | let greeting = "hello world!";

This is:
  payloads (defined as array jsPayload)
But somewhere wanted:
  allAges (defined as array ages)

The incompatible parts:
  jsPayload (defined as Js.t {• age : int, name : string })
  vs
  ages (defined as array int)
```

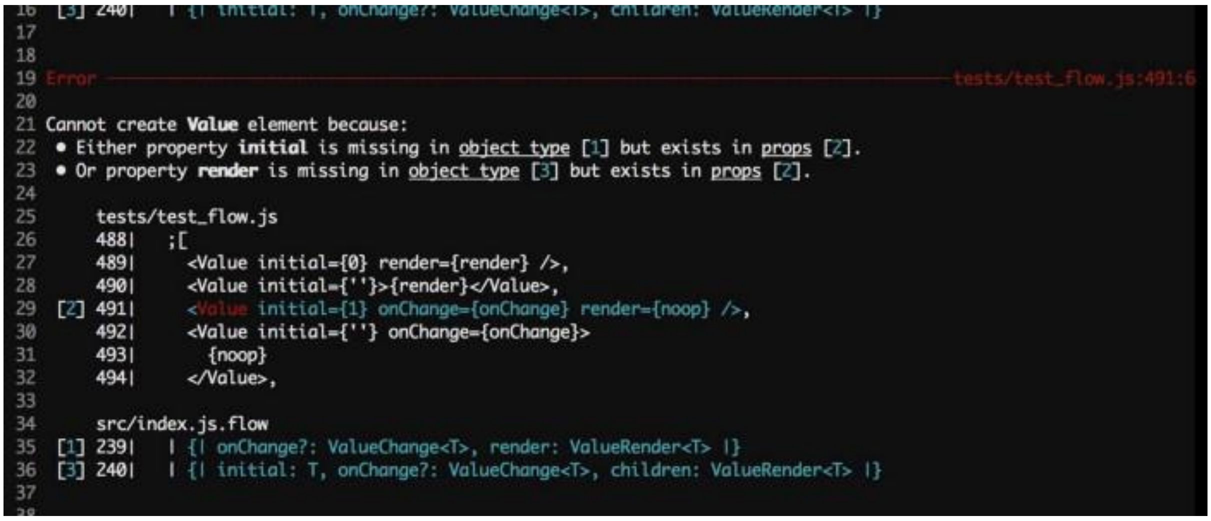
The changes are mostly in formatting and wording rather than in fundamentally changing the information presented to the user, but it makes a *huge* difference in how readable and friendly the message is. The structure and colours given to the error makes it easier to parse by eye and lower the cognitive load, and the more positive wording makes the error more friendly.

[11] Way, Way, Waaaay Nicer Error Messages!

Flow

The authors of Flow have also done a lot of work on improving their error messages, and have written a blog post on the changes[12] and on some general guidelines for compiler errors[13].

Here's an example:



```
[3] 240|    | {| initial: T, onChange?: ValueChange<T>, children: ValueRender<T> |}


Error ------------------------------------------------------------ tests/test_flow.js:491:6

Cannot create Value element because:
 • Either property initial is missing in object type [1] but exists in props [2].
 • Or property render is missing in object type [3] but exists in props [2].

     tests/test_flow.js
     488|   ;[
     489|     <Value initial={0} render={render} />,
     490|     <Value initial={''}>{render}</Value>,
 [2] 491|     <Value initial={1} onChange={onChange} render={noop} />,
     492|     <Value initial={''} onChange={onChange}>
     493|       {noop}
     494|     </Value>,

     src/index.js.flow
 [1] 239|    | {| onChange?: ValueChange<T>, render: ValueRender<T> |}
 [3] 240|    | {| initial: T, onChange?: ValueChange<T>, children: ValueRender<T> |}
```

Note that a description of the error is given in plain English, and the disjunction is plainly surfaced in the formatting.

A couple of guidelines from their post which I think are interesting:

> Write messages in first person plural. That is, use "we". For example "we see an error". This personifies the compiler as a *team* of people looking for bugs in the developer's code. By personifying our type checker error messages feel like a dialogue. Elm's error messages are famous for using first person: "I see an error". First person feels a bit uncomfortable to me. A compiler is certainly not a single person, nor is it built by a single person. I prefer "we" as a compromise.
>
> Use present tense instead of past tense. Instead of "we found" say "we see". When an error is displayed to a user, the code is currently in a bad state. From the compiler author's perspective, the compiler runs at discrete points in time and "finds" errors at those points. From the developer's perspective an error in their IDE reflects the current state of the program, not a discrete compiler run. Prefer the developer's IDE context, use present tense.

D

D has a ranges feature not unlike C++20 ranges. Here's the declaration for one of the functions with a more complex constraint:

```
auto chain(Ranges...)(Ranges rs)
if (Ranges.length > 0 && allSatisfy!(isInputRange, staticMap!(Unqual,
Ranges)) && !is(CommonType!(staticMap!(ElementType, staticMap!(Unqual,
Ranges))) == void));
```

This constraint checks that you pass at least one range, that all the ranges are input ranges, and that there's a common ElementType (which I assume is like value_type in C++) between them all.

[12] Better Flow Error Messages for the JavaScript Ecosystem
[13] Writing Good Compiler Error Messages

If we give chain something that is *not* an input range, like an integer.

```
int[] arr = [ 1, 2, 3, 4 ];
auto s = chain(arr, 3);
```

Then we get this error message:

```
Error: template `std.range.chain` cannot deduce function from argument
types `!()(int[], int)`, candidates are:
/dlang/dmd/linux/bin64/../../src/phobos/std/range/package.d(887):
`chain(Ranges...)(Ranges rs)`
  with `Ranges = (int[], int)`
  must satisfy the following constraint:
`       allSatisfy!(isInputRange, staticMap!(Unqual, Ranges))`
```

This error narrowed down which constraint in the conjunction failed (i.e. it wasn't the length check or the common type check), but it didn't tell us *which* argument failed the constraint, and it didn't tell us *why*, or what we can do about it.

C#

Here's a C# example of passing an argument that doesn't satisfy the constraint:

```
    class Program
    {
        class a
        {
            public static void CallMe<T>(T s) where T : struct { }
        }

        static void Main(string[] args)
        {
            a.CallMe("hello");
        }
    }
```

We get an error message on the call to CallMe:

The type 'string' must be a non-nullable value type in order to use it as parameter 'T' in the generic type or method 'Program.a.CallMe<T>(T)'

No fixes are offered (which is reasonable). A fix could be offered to generate an overload with a string parameter or to remove the constraint (which may give hints to the problem but are unlikely what is intended)

Rust

Consider this Rust program:

```
fn main() {
    let mut a = [0, 1, 2].iter().intersperse(&100);

    // Partition in-place between evens and odds
    let i = a.into_iter().partition_in_place(|&n| n % 2 == 0);
}
```

There's a couple of issues with this:

1. `partition_in_place` requires a `DoubleEndedIterator`, which `Intersperse` is not.

2. `partition_in_place` needs to be able to mutate the underlying iterator, which it cannot, because `Intersperse` doesn't support it.

rustc diagnoses both issues in a clear, concise manner:

```
error[E0277]: the trait bound `Intersperse<std::slice::Iter<'_, {integer}>>:
DoubleEndedIterator` is not satisfied

--> <source>:8:27

 |

8 |    let i = a.into_iter().partition_in_place(|&n| n % 2 == 0);

 |                         ^^^^^^^^^^^^^^^^^^ the trait `DoubleEndedIterator`
is not implemented for `Intersperse<std::slice::Iter<'_, {integer}>>`



error[E0271]: type mismatch resolving `<Intersperse<std::slice::Iter<'_,
{integer}>> as Iterator>::Item == &mut _`

--> <source>:8:27

 |

8 |    let i = a.into_iter().partition_in_place(|&n| n % 2 == 0);

 |                         ^^^^^^^^^^^^^^^^^^ types differ in mutability

 |

 = note:      expected reference `&{integer}`

         found mutable reference `&mut _`

error: aborting due to 2 previous errors

Some errors have detailed explanations: E0271, E0277.

For more information about an error, try `rustc --explain E0271`.
```

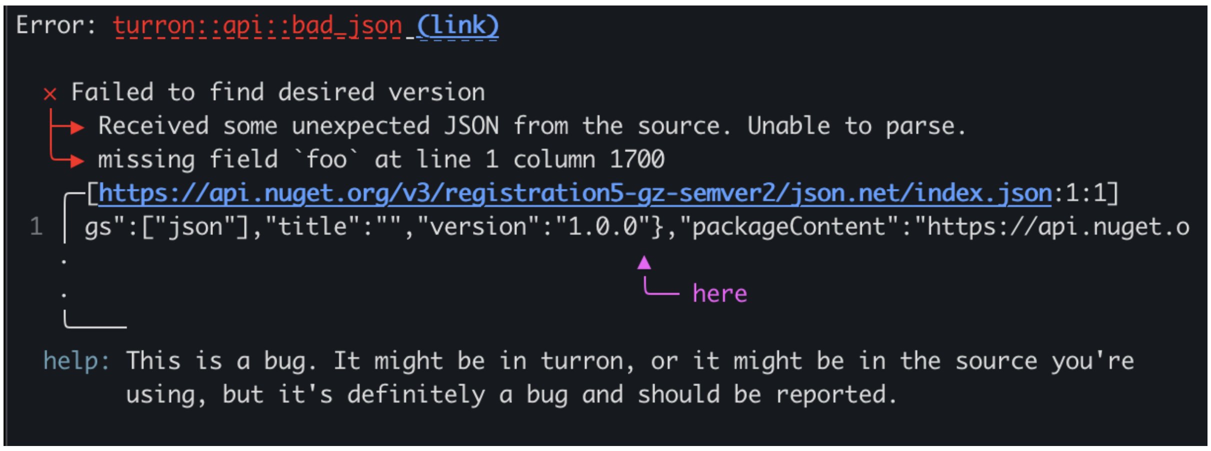
Note those last two lines, which give the user the potential for interactivity to get more detailed error descriptions. If we follow the compiler's advice and pass that error message, then we get a very detailed description of what it means for a type to mismatch an associated type of a trait. It's too long to reproduce here, but you can also find it online[14].

A couple of relevant documents produced by the Rust team include their guide to writing error messages and their long error code normalization RFC. The former includes this colourful definition of what it means for an error message to be readable:

[14] E0271 in the Rust Compiler Error Index

> Write in plain simple English. If your message, when shown on a – possibly small – screen (which hasn't been cleaned for a while), cannot be understood by a normal programmer, who just came out of bed after a night partying, it's too complex.

Another example of Rust excelling in diagnostics is the miette library, which generates very beautiful errors:



```
Error: turron::api::bad_json (link)

  × Failed to find desired version
  ├─▶ Received some unexpected JSON from the source. Unable to parse.
  ╰─▶ missing field `foo` at line 1 column 1700
   ╭─[https://api.nuget.org/v3/registration5-gz-semver2/json.net/index.json:1:1]
 1 │ gs":["json"],"title":"","version":"1.0.0"},"packageContent":"https://api.nuget.o
   ·                                             ▲
   ·                                             ╰── here
   ╰────
  help: This is a bug. It might be in turron, or it might be in the source you're
        using, but it's definitely a bug and should be reported.
```

Current State and Possible Designs

Note that all diagnostics captured here were generated in August 2021 and may not fully represent the current state of the tooling.

Example 1

Consider this code, where we pass an argument to a function template which doesn't satisfy its constraints.

```
template <std::integral N>
void f(N n);

int main() {
    f(nullptr);
}
```

The compiler error generated is:

```
<source>(9): error C2672: 'f': no matching overloaded function found
<source>(9): error C7602: 'f': the associated constraints are not satisfied
<source>(6): note: see declaration of 'f'
```

How does this compare against our guidelines?

Readability is *okay*. "Associated constraints" is a very useful term to connect concepts (c.f. scaffolding), but it's also jargony, which can reduce readability. It does *not* tell us what the constraint was (context, which would also help clarify the jargon term), or give us any solutions/hints. The tone is also quite negative.

This is the error which GCC generates for the same code:

```
<source>: In function 'int main()':
<source>:9:6: error: no matching function for call to 'f(std::nullptr_t)'
    9 |     f(nullptr);
```

```
      |     ~^~~~~~~~~
<source>:6:6: note: candidate: 'template<class N>  requires  integral<N> void
f(N)'
    6 | void f(N n);
      |      ^
<source>:6:6: note:   template argument deduction/substitution failed:
<source>:6:6: note: constraints not satisfied
In file included from /opt/compiler-explorer/gcc-
11.1.0/include/c++/11.1.0/ranges:37,
                 from <source>:1:
/opt/compiler-explorer/gcc-11.1.0/include/c++/11.1.0/concepts: In substitution
of 'template<class N>  requires  integral<N> void f(N) [with N =
std::nullptr_t]':
<source>:9:6:   required from here
/opt/compiler-explorer/gcc-11.1.0/include/c++/11.1.0/concepts:102:13:
required for the satisfaction of 'integral<N>' [with N = std::nullptr_t]
/opt/compiler-explorer/gcc-11.1.0/include/c++/11.1.0/concepts:102:24: note:
the expression 'is_integral_v<_Tp> [with _Tp = std::nullptr_t]' evaluated to
'false'
  102 |     concept integral = is_integral_v<_Tp>;
      |                        ^~~~~~~~~~~~~~~~~~
```

This gives us a *lot* of context. Too much, I would argue. I don't think most users need to know which line of code in a standard library header the instantiation of `std::integral` failed at. I also don't think they need to know that the `integral` concept is implemented in terms of a type trait.

I think GCC is commendable in giving the user all they need to diagnose the error, but the design of it raises cognitive load and hurts readability.

So what *should* this error look like? Here's one possibility:

```
In function 'int main()' [<source>]:
Error: We can't find a matching function for a call to 'f(std::nullptr_t)' at
<source>:9:6:
    9 |     f(nullptr);
      |     ~^~~~~~~~~

We found 1 candidate:
  Candidate at <source>:9:6:
      6 | void f(N n);
        |      ^
    The candidate was not valid because its constraints were not satisfied:
      5 | template <std::integral N> [with N = std::nullptr_t]
        |           ^~~~~~~~~~~~~~~
      6 | void f(N n);
    Hint: 'integral<T>' is only satisfied if 'T' is an integral type, see
<https://en.cppreference.com/w/cpp/types/is_integral>
```

I think this gives the right balance between providing necessary context while maintaining readability, lowering cognitive load, and providing relevant hints, all in a positive tone.

Some changes I made:

- Rephrased the error in terms of "we can't find a matching function" rather than "no matching function".
- Moved line/column numbers to the end of lines so that they're not distracting from the useful information.
- Explicitly stated the number of candidates found.
- Provided an English-language description of the concept which failed. Since the concepts in the standard library are well-defined and not too large in number, I think this would be a reasonable and valuable thing for a compiler to do.
- Provided a link to go to for more information.
- Did not diagnose standard library internals.
- Added colours and text weighting, which would require IDE changes to be displayed correctly (URLs should also be made clickable).

For comparison, here is Clang's error:

```
<source>:7:5: error: no matching function for call to 'f'

   f(nullptr);

   ^

<source>:4:6: note: candidate template ignored: constraints not satisfied [with N = nullptr_t]

void f(N n);

    ^

<source>:3:16: note: because 'nullptr_t' does not satisfy 'integral'

template <std::integral N>

              ^
```

This is fairly similar to the level of detail provided in my version, but lacks the stating of number of candidates found, link, positive phrasing, and English-language description of the concept.

Example 2

Consider this example, which attempts to pass an rvalue container into a range adaptor, which should fail because rvalue containers do not satisfy `borrowed_range`.

```
std::vector<int> get_vec();

int main() {
    for(auto i : std::views::drop(get_vec(), 1)) {
        std::cout << i;
    }
}
```

Here is the error which MSVC outputs:

```
<source>(8): error C2672: 'operator __surrogate_func': no matching overloaded
function found
```

```
<source>(8): error C7602: 'std::ranges::views::_Drop_fn::operator ()': the
associated constraints are not satisfied
C:/data/msvc/14.29.29917-Pre/include\ranges(2094): note: see declaration of
'std::ranges::views::_Drop_fn::operator ()'
<source>(8): error C2780: 'auto std::ranges::views::_Drop_fn::operator ()(_Ty)
noexcept const': expects 1 arguments - 2 provided
C:/data/msvc/14.29.29917-Pre/include\ranges(2112): note: see declaration of
'std::ranges::views::_Drop_fn::operator ()'
<source>(8): error C3531: 'i': a symbol whose type contains 'auto' must have
an initializer
<source>(8): error C2143: syntax error: missing ';' before ':'
<source>(8): error C7602: 'std::ranges::views::_Drop_fn::operator ()': the
associated constraints are not satisfied
C:/data/msvc/14.29.29917-Pre/include\ranges(2094): note: see declaration of
'std::ranges::views::_Drop_fn::operator ()'
<source>(8): error C2143: syntax error: missing ';' before ')'
```

Readability is impaired by the `__surrogate_func`, which is an implementation detail of `views::drop`, and by the internal header path. The error does tell us that trying to call the `operator()` failed due to unsatisfied constraints, but it doesn't tell us what those constraints are, or why they failed. It also contains a couple of spurious failures (the syntax errors) and a duplicated diagnostic (one of the things that raises cognitive load).

Here is GCC's output:

```
<source>: In function 'int main()':
<source>:8:34: error: no match for call to '(const std::ranges::views::_Drop)
(std::vector<int>, int)'
    8 |     for(auto i : std::views::drop(get_vec(), 1)) {
      |                  ~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~
In file included from <source>:1:
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:2330:9:
note: candidate: 'template<class _Range, class _Tp>  requires
(viewable_range<_Range>) && (__can_drop_view<_Range, _Tp>) constexpr auto
std::ranges::views::_Drop::operator()(_Range&&, _Tp&&) const'
 2330 |         operator()(_Range&& __r, _Tp&& __n) const
      |         ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:2330:9:
note:   template argument deduction/substitution failed:
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:2330:9:
note: constraints not satisfied
In file included from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/string_view:44,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/basic_string.h:48,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/string:55,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/streambuf:41,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/streambuf_iterator.h:35,
```

```
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/iterator:66,
                 from /opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/ranges:43,
                 from <source>:1:
<source>: In substitution of 'template<class _Range, class _Tp>  requires
(viewable_range<_Range>) && (__can_drop_view<_Range, _Tp>) constexpr auto
std::ranges::views::_Drop::operator()(_Range&&, _Tp&&) const [with _Range =
std::vector<int>; _Tp = int]':
<source>:8:34:   required from here
/opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/ranges_base.h:669:13:   required for the
satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210614/include/c++/12.0.0/bits/ranges_base.h:670:31: note: no operand of the
disjunction is satisfied
  670 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |           ~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1plus: note: set '-fconcepts-diagnostics-depth=' to at least 2 for more
detail
```

Similar to the last example, GCC gives us a lot of useful information, but it's hard to read between all the standard library internals and general syntactic noise. It *does* tell us that one of `std::borrowed_range` and `std::view` failed, and that to get more information we can pass a compiler switch.

If we pass that switch then we get:

```
<source>: In function 'int main()':
<source>:7:34: error: no match for call to '(const std::ranges::views::_Drop)
(std::vector<int>, int)'
    7 |     for(auto i : std::views::drop(get_vec(), 1)) {
      |                  ~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~
In file included from <source>:2:
/opt/compiler-explorer/gcc-trunk-20210708/include/c++/12.0.0/ranges:2358:9:
note: candidate: 'template<class _Range, class _Tp>  requires
(viewable_range<_Range>) && (__can_drop_view<_Range, _Tp>) constexpr auto
std::ranges::views::_Drop::operator()(_Range&&, _Tp&&) const'
 2358 |         operator()(_Range&& __r, _Tp&& __n) const
      |         ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210708/include/c++/12.0.0/ranges:2358:9:
note:   template argument deduction/substitution failed:
/opt/compiler-explorer/gcc-trunk-20210708/include/c++/12.0.0/ranges:2358:9:
note: constraints not satisfied
In file included from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/string_view:44,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/basic_string.h:48,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/string:55,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/locale_classes.h:40,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ios_base.h:41,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/streambuf:41,
```

```
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/streambuf_iterator.h:35,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/iterator:66,
                 from /opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/ranges:43,
                 from <source>:2:
<source>: In substitution of 'template<class _Range, class _Tp>  requires
(viewable_range<_Range>) && (__can_drop_view<_Range, _Tp>) constexpr auto
std::ranges::views::_Drop::operator()(_Range&&, _Tp&&) const [with _Range =
std::vector<int>; _Tp = int]':
<source>:7:34:   required from here
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:666:13:   required for the
satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:667:31: note: no operand of the
disjunction is satisfied
  667 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |           ~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:667:11: note: the operand
'borrowed_range<_Tp>' is unsatisfied because
  667 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |           ~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:83:15:   required for the
satisfaction of '__maybe_borrowed_range<_Tp>' [with _Tp = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:582:13:   required for the
satisfaction of 'borrowed_range<_Tp>' [with _Tp = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:666:13:   required for the
satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:85:11: note: no operand of the
disjunction is satisfied
   84 |         = is_lvalue_reference_v<_Tp>
      |           ~~~~~~~~~~~~~~~~~~~~~~~~~~
   85 |           || enable_borrowed_range<remove_cvref_t<_Tp>>;
      |           ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:667:34: note: the operand
'view<typename std::remove_cvref<_Tp>::type>' is unsatisfied because
  667 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |           ~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:621:13:   required for the
satisfaction of 'view<typename std::remove_cvref<_Tp>::type>' [with _Tp =
std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:666:13:   required for the
```

```
satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int,
std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-
20210708/include/c++/12.0.0/bits/ranges_base.h:622:39: note: the expression
'enable_view<_Tp> [with _Tp = std::vector<int, std::allocator<int> >]'
evaluated to 'false'
  622 |       = range<_Tp> && movable<_Tp> && enable_view<_Tp>;
      |                                       ^~~~~~~~~~~~~~~~
cc1plus: note: set '-fconcepts-diagnostics-depth=' to at least 3 for more
detail
```

Again, this is very hard to read, but it *does* tell us what the actual problem was: that the argument is not an lvalue reference. Unfortunately, extracting this information from the error requires a fair bit of background knowledge. If the compiler could output a specific error for `std::borrowed_range` then it would be a lot more clear. Something like:

```
note: the operand 'borrowed_range<_Tp>' is unsatisfied because
  667 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);

note: _Tp is not an lvalue reference and does not model std::view
<https://en.cppreference.com/w/cpp/ranges/view>. [with _Tp = std::vector<int,
std::allocator<int> >]
```

Perhaps the compiler could even recognise the pattern of using the prvalue result of a function call for something that requires `std::borrowed_range` and issue a note suggesting to store the result in a local variable.

For comparison, here is Clang's error (using libstdc++'s ranges implementation), which is fairly similar to GCC's:

```
<source>:8:18: error: no matching function for call to object of type 'const
std::ranges::views::_Drop'

   for(auto i : std::views::drop(get_vec(), 1)) {

                ^~~~~~~~~~~~~~~~

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:2361:2: note: candidate
template ignored: constraints not satisfied [with _Range = std::vector<int>,
_Tp = int]

       operator() [[nodiscard]] (_Range&& __r, _Tp&& __n) const

       ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:2358:16: note: because
'std::vector<int>' does not satisfy 'viewable_range'

     template<viewable_range _Range, typename _Tp>

              ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:672:11: note:
because 'std::vector<int>' does not satisfy 'borrowed_range'
```

```
     && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
         ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:588:33: note:
because 'std::vector<int>' does not satisfy '__maybe_borrowed_range'
     = range<_Tp> && __detail::__maybe_borrowed_range<_Tp>;
                               ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:84:4: note:
because 'is_lvalue_reference_v<std::vector<int> >' evaluated to false
       = is_lvalue_reference_v<_Tp>
         ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:85:7: note: and
'enable_borrowed_range<remove_cvref_t<std::vector<int> > >' evaluated to false
         || enable_borrowed_range<remove_cvref_t<_Tp>>;
            ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:672:34: note: and
'remove_cvref_t<std::vector<int>>' (aka 'std::vector<int>') does not satisfy
'view'
     && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
                                ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ranges_base.h:627:39: note:
because 'enable_view<std::vector<int> >' evaluated to false
     = range<_Tp> && movable<_Tp> && enable_view<_Tp>;
                                     ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:844:2: note: candidate
template ignored: constraints not satisfied [with _Args = <std::vector<int>,
int>]
       operator()(_Args&&... __args) const
       ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:842:11: note: because
'__adaptor_partial_app_viable<std::ranges::views::_Drop, std::vector<int>,
int>' evaluated to false
       requires __adaptor_partial_app_viable<_Derived, _Args...>
```

```
                  ^
/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:791:11: note: because
'sizeof...(_Args) == _Drop::_S_arity - 1' (2 == 1) evaluated to false

     && (sizeof...(_Args) == _Adaptor::_S_arity - 1)
         ^

1 error generated.
```

Example 3

Consider this very similar example, which uses the partial application and overloaded pipe operator syntax of Ranges:

```
std::vector<int> get_vec();

int main() {
    for(auto i : get_vec() | std::views::drop(1)) {
        std::cout << i;
    }
}
```

Here's the error from MSVC:

```
<source>(8): error C2678: binary '|': no operator found which takes a left-
hand operand of type 'std::vector<int,std::allocator<int>>' (or there is no
acceptable conversion)
C:/data/msvc/14.29.29917-Pre/include\cstddef(43): note: could be 'std::byte
std::operator |(const std::byte,const std::byte) noexcept' [found using
argument-dependent lookup]
<source>(8): note: while trying to match the argument list
'(std::vector<int,std::allocator<int>>,
std::ranges::views::_Drop_fn::_Partial<_Ty>)'
        with
        [
            _Ty=int
        ]
<source>(8): error C3531: 'i': a symbol whose type contains 'auto' must have
an initializer
<source>(8): error C2143: syntax error: missing ';' before ':'
<source>(8): error C2678: binary '|': no operator found which takes a left-
hand operand of type 'std::vector<int,std::allocator<int>>' (or there is no
acceptable conversion)
C:/data/msvc/14.29.29917-Pre/include\cstddef(43): note: could be 'std::byte
std::operator |(const std::byte,const std::byte) noexcept' [found using
argument-dependent lookup]
<source>(8): note: while trying to match the argument list
'(std::vector<int,std::allocator<int>>,
std::ranges::views::_Drop_fn::_Partial<_Ty>)'
        with
        [
            _Ty=int
        ]
<source>(8): error C2143: syntax error: missing ';' before ')'
```

This error gives essentially no useful information that could be used to fix the error. It gives a couple of candidate overloads, but neither is the one we want to call. The error also causes two cascading failures which have nothing to do with the overload resolution failure (the missing initializer and missing semicolon).

Here's GCC's error:

```
<source>: In function 'int main()':
<source>:8:28: error: no match for 'operator|' (operand types are
'std::vector<int>' and
'std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>')
    8 |     for(auto i : get_vec() | std::views::drop(1)) {
      |                  ~~~~~~~~~ ^ ~~~~~~~~~~~~~~~~~~~
      |                         |                    |
      |                         std::vector<int>
std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>
```

```
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:774:7:
note: candidate: 'template<class _Self, class _Range>  requires
(derived_from<typename std::remove_cvref<_Tp>::type,
std::ranges::views::__adaptor::_RangeAdaptorClosure>) &&
(__adaptor_invocable<_Self, _Range>) constexpr auto
std::ranges::views::__adaptor::operator|(_Range&&, _Self&&)'
  774 |       operator|(_Range&& __r, _Self&& __self)
      |       ^~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:774:7:
note:   template argument deduction/substitution failed:
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:774:7:
note: constraints not satisfied
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges: In
substitution of 'template<class _Self, class _Range>  requires
(derived_from<typename std::remove_cvref<_Tp>::type,
std::ranges::views::__adaptor::_RangeAdaptorClosure>) &&
(__adaptor_invocable<_Self, _Range>) constexpr auto
std::ranges::views::__adaptor::operator|(_Range&&, _Self&&) [with _Self =
std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>;
_Range = std::vector<int>]':
<source>:8:48:   required from here
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:746:13:
required for the satisfaction of '__adaptor_invocable<_Self, _Range>' [with
_Self = std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop,
int>; _Range = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:747:9:
in requirements  [with _Args = {std::vector<int, std::allocator<int> >};
_Adaptor = std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop,
int>]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/ranges:747:44:
note: the required expression 'declval<_Adaptor>()((declval<_Args>)()...)' is
invalid
  747 |       = requires { std::declval<_Adaptor>()(declval<_Args>()...); };
      |                    ~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~
cc1plus: note: set '-fconcepts-diagnostics-depth=' to at least 2 for more
detail
```

Again, this is a ton of information and full of complex standard library symbols and implementation details. It also doesn't tell us what the problem is (that `borrowed_range` failed).

At least the compiler does allow some dynamic interaction by letting us supply `-fconcepts-diagnostics-depth` to get more information. If we supply the value 4 for that flag, then there's a *ton* more output, more than I'd reasonably want to put in this document. Somewhere in there is the relevant error:

```
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:669:13:   required for the satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:670:31: note: no operand of the disjunction is satisfied
  670 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |          ~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:670:11: note: the operand 'borrowed_range<_Tp>' is unsatisfied because
  670 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |          ~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:83:15:   required for the satisfaction of '__maybe_borrowed_range<_Tp>' [with _Tp = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:584:13:   required for the satisfaction of 'borrowed_range<_Tp>' [with _Tp = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:669:13:   required for the satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:85:11: note: no operand of the disjunction is satisfied
   84 |         = is_lvalue_reference_v<_Tp>
      |           ~~~~~~~~~~~~~~~~~~~~~~~~~~
   85 |           || enable_borrowed_range<remove_cvref_t<_Tp>>;
      |           ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:670:34: note: the operand 'view<typename std::remove_cvref<_Tp>::type>' is unsatisfied because
  670 |       && (borrowed_range<_Tp> || view<remove_cvref_t<_Tp>>);
      |          ~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:623:13:   required for the satisfaction of 'view<typename std::remove_cvref<_Tp>::type>' [with _Tp = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:669:13:   required for the satisfaction of 'viewable_range<_Range>' [with _Range = std::vector<int, std::allocator<int> >]
/opt/compiler-explorer/gcc-trunk-20210614/include/c++/12.0.0/bits/ranges_base.h:625:12: note: the expression
```

```
'enable_view<_Tp> [with _Tp = std::vector<int, std::allocator<int> >]'
evaluated to 'false'
  625 |          && enable_view<_Tp>;
      |             ^~~~~~~~~~~~~~~~
```

Similar to the first example, we got all the information we could possibly need to diagnose the issue, but it's really hard to comprehend.

Generating a good error for this is complicated by two main points:

- The compiler can't know which `operator|` the user meant to call.
- The compiler can't know which constraint failure is the relevant one.

Of course, it could use some heuristics, such as recognizing that the right hand side is a range adaptor, so the user probably meant to use the `operator|` which pipes a range into a range adaptor. It could also recognize the pattern of piping a prvalue container into a range adaptor and present a specialised error message, but that'd likely only work for standard library containers.

Again, Clang's diagnostic for comparison:

```
<source>:8:28: error: invalid operands to binary expression
('std::vector<int>' and
'std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>')

   for(auto i : get_vec() | std::views::drop(1)) {

                ~~~~~~~~~ ^ ~~~~~~~~~~~~~~~~~~~

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ios_base.h:87:3: note:
candidate function not viable: no known conversion from 'std::vector<int>' to
'std::_Ios_Fmtflags' for 1st argument

 operator|(_Ios_Fmtflags __a, _Ios_Fmtflags __b)

 ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ios_base.h:129:3: note:
candidate function not viable: no known conversion from 'std::vector<int>' to
'std::_Ios_Openmode' for 1st argument

 operator|(_Ios_Openmode __a, _Ios_Openmode __b)

 ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/bits/ios_base.h:169:3: note:
candidate function not viable: no known conversion from 'std::vector<int>' to
'std::_Ios_Iostate' for 1st argument

 operator|(_Ios_Iostate __a, _Ios_Iostate __b)

 ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:812:7: note: candidate
template ignored: constraints not satisfied [with _Self =
```

```
std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>,
_Range = std::vector<int>]

     operator|(_Range&& __r, _Self&& __self)

     ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:810:5: note: because
'__adaptor_invocable<std::ranges::views::__adaptor::_Partial<std::ranges::view
s::_Drop, int>, std::vector<int> >' evaluated to false

       && __adaptor_invocable<_Self, _Range>

          ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:785:20: note: because
'std::declval<_Adaptor>()(declval<_Args>()...)' would be invalid: no matching
function for call to object of type
'std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop, int>'

     = requires { std::declval<_Adaptor>()(declval<_Args>()...); };

                  ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:821:7: note: candidate
template ignored: constraints not satisfied [with _Lhs = std::vector<int>,
_Rhs = std::ranges::views::__adaptor::_Partial<std::ranges::views::_Drop,
int>]

     operator|(_Lhs __lhs, _Rhs __rhs)

     ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/ranges:818:16: note: because
'derived_from<std::vector<int>,
std::ranges::views::__adaptor::_RangeAdaptorClosure>' evaluated to false

     requires derived_from<_Lhs, _RangeAdaptorClosure>

              ^

/opt/compiler-explorer/gcc-snapshot/lib/gcc/x86_64-linux-
gnu/12.0.0/../../../../include/c++/12.0.0/concepts:67:28: note: because
'__is_base_of(std::ranges::views::__adaptor::_RangeAdaptorClosure,
std::vector<int>)' evaluated to false

   concept derived_from = __is_base_of(_Base, _Derived)

                          ^

1 error generated.
```

This time Clang doesn't give enough information: it tells us that the correct overload had a constraint failure inside some standard library machinery, but it doesn't get deep enough to show why it failed.

Future Direction

This section contains some options of the directions in which tool vendors could take our diagnostics.

Diagnostics Language

We can make changes to the language we use in our errors in aid of readability, positive tone use. We could rework diagnostics to be in full sentences and to use first-person pronouns (I or we), taking direction from other modern language's developments. We should always keep in mind the friendliness of our diagnostics, and whether they could be off-putting.

Context

Currently MSVC does not provide helpful context when a constraint failure occurs, particularly in the case of multiple overloads.

Reiterating from Context, for concepts-related diagnostics, I think the most important pieces of context that can be given to the user are:

1. Which constraints failed and why.
2. If there were multiple candidates considered, what were they and why were they discarded.

This is what GCC does today. It gives you all of the context you need, although it gives you too much, and visualises it in a way which is hard to read.

Just My Code

A problem that GCC's output has is that its constraint failures often go deep into some standard library headers, whose code is hard to understand, and is difficult to connect to the user code. Ideally we would minimise this, potentially by only diagnosing the user code and specifically handling each standard library concept to avoid needing to rely on STL internals to provide necessary context.

Standard Library Symbol Names

One of the most common complaints about C++ compiler errors is the long, complex standard library names which appear in diagnostics. As we add more complex diagnostics which will necessarily involve these types, special care should be given to how readable the output is.

Natural Language Concepts Descriptions

Since the set of standard concepts is not too large, it may be possible to output descriptions of concepts in natural language when there is a constraint failure. For example, `borrowed_range` is a fairly complex concept (both the C++ and English versions of "concept"). If constraint failure diagnostics involving it included a natural language description of what a borrowed range, and potentially examples, this could help make the error more friendly and comprehensible.

External Links

Other languages such as Elm will output links to online resources that can be accessed for more information. We should consider doing the same, especially if we adopt specific natural language descriptions for concepts.

Visualisation

So far I've mostly discussed purely textual output. However, there are ways we could visualise the output to make it more digestible.

We could make overload resolution failures in IDE error lists expand into a list of candidates, so it's easier to pinpoint which you wanted to call and work out what failed for that candidate. For example, the error list could begin in this state (text is just for illustrative purposes):

- Call <x> failed, no candidate found

Then when the error is clicked on, it expands to:

- Call <x> failed, no candidate found
    - Candidate 1 signature
    - Candidate 2 signature
    - Candidate 3 signature

You could then click a specific candidate to find out why it failed:

- Call <x> failed, no candidate found
    - Candidate 1 signature
    - Candidate 2 signature
        - Failed to instantiate because <y>
    - Candidate 3 signature

Additionally, we could visualise constraint failures as trees, where disjunctions and conjunctions appear as forks in the tree, and you can expand nodes to delve deeper into the reasons that they failed. E.g.:

- Instantiation of function <x> failed:
    - Conjunction constraint <a> failed because all of the following failed
        - Disjunction constraint <b> failed because none of the following were true
            - Requires clause <c> failed because […]
            - Type trait <d> failed
        - Requires clause <e> failed because […]

All of this would help manage cognitive load and provide dynamic interaction.

Structured Output

Currently, most tools print their output to the console or error window as text. However, they could instead output structured output such as JSON, XML, or formats designed for structured errors such as SARIF. This would allow tools to filter, manipulate, and visualise the errors in any way they wanted.

For example, a tool could generate an HTML web page from the error output and load it into your browser. Or one specialised for helping with template errors could analyse the users code and provide further assistance in fixing the errors.

Conclusion

In the above I:

- summarized the state of the art in compiler error message research,
- gave examples of what production compilers are doing to improve their error messages,
- presented the current state of error messages for concepts-based C++ programming in MSVC, Clang and GCC, and
- suggested potential improvements to the state of the art.

I hope that this paper can start some discussion on what direction compiler errors should be moving in for C++.