Concepts C++20?

Mikel Irazabal

mikel@irazabal.eu

June 28, 2018

History of Templates

They tried:

- Full generality/expressiveness
- Zero overhead compared to hand coding
- Well-specified interface

They were able to achieve:

- Turing completeness
- Better than hand-coding performance
- Lousy interfaces (basically compile-time duck typing)

Duck Typing

- ”If it walks like a duck and it quacks like a duck, then it must be a duck.”

History

Example (Compile-time Duck Typing in python)

```
def all(seq, pred):
    for x in seq:
        if not pred(x):
            return False;
    return True;

def test(seq, fn): print all(seq, fn)

def is_even(n): return n % 2 == 0
def is_lower(c): return 'a' <= c and c <= 'z'
def is_conmutative(x,y,op):
    temp1 =  op(x,y)
    temp2 = op(y,x)
    return temp1 == temp2
```

History

Example (Compile-time Duck Typing in python)

```
def main():
    test([0,2], is_even)
    test("abC", is_lower)
    test([0,2 ],is_conmutative)

if __name__ == "__main__":
    main()
```

History

Example (Compile-time Duck Typing in python)

```
True
False
Traceback (most recent call last):
  File "test.py", line 27, in <module>
    main()
  File "test.py", line 24, in main
    test([0,2 ],is_conmutative)
  File "test.py", line 8, in test
    print all(seq, fn)
  File "test.py", line 3, in all
    if not pred(x):
TypeError: is_conmutative() takes exactly 3 arguments (1 give
```

History

Example (Compile-time Duck Typing for std::sort)

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );

template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy, RandomIt first,
           RandomIt last );

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );

template< class ExecutionPolicy, class RandomIt, class Compare
void sort( ExecutionPolicy&& policy, RandomIt first,
           RandomIt last, Compare comp );

\begin{frame}
```

Random Access Iterator

Example (Simple sort programm)

```
#include <algorithm>
#include <list>

int main()
{
  std::list<int> l_{ 1,3,5,7,9,7,5,3,1};
  std::sort(std::begin(l_), std::end(l_) );

  return 0;
}
```

Random Access Iterator

Example (Simple sort programm)

```
clang++ sort_list.cpp -std=c++11

In file included from sort_list.cpp:1:
In file included from /usr/bin/../lib/gcc/x86_64-linux-gnu/
5.4.0/../../../../include/c++/5.4.0/algorithm:62:
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
/5.4.0/bits/stl_algo.h:1964:22: error: invalid operand
s to binary expression ('std::_List_iterator<int>' and
 'std::_List_iterator<int>')
                std::__lg(__last - __first) * 2,
                                          ~~~~~~~ ^ ~~~~~~~~

/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../
../include/c++/5.4.0/bits/stl_algo.h:4698:12: note:
in instantiation of function template specialization
'std::__sort<std::_List_iterator<int>,
```

Random Access Iterator

Example (Simple sort programm)

```
      __gnu_cxx::__ops::_Iter_less_iter>'
       requested here
      std::__sort(__first, __last, __gnu_cxx::__ops::__iter_le
           ^
sort_list.cpp:6:7: note: in instantiation of function
template specialization
'std::sort<std::_List_iterator<int> >'
requested here
        std::sort(std::begin(l), std::end(l) );
             ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/
../../../../include/c++/5.4.0/bits/
stl_bvector.h:208:3: note: candidate function
not viable: no known conversion from
'std::_List_iterator<int>' to
```

Random Access Iterator

Example (Simple sort programm)

```
      'const std::_Bit_iterator_base'
      for 1st argument
  operator-(const _Bit_iterator_base& __x,
   const _Bit_iterator_base& __y)
  ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/..
/../../../include/c++/5.4.0/bits/
stl_iterator.h:328:5: note: candid
ate template ignored: could not match
'reverse_iterator' against '_List_iterator'
    operator-(const reverse_iterator<_Iterator>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/..
/../../../include/c++/5.4.0/bits/
stl_iterator.h:380:5: note: candidate template
ignored: could not match 'reverse_iterator'
```

Random Access Iterator

Example (Simple sort programm)

```
stl_iterator.h:380:5: note: candidate template
 ignored: could not match ’reverse_iterator’
 against ’_List_iterator’
    operator-(const reverse_iterator<_IteratorL>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../..
/../../include/c++/5.4.0/bits/stl_iterator.h:1138:5:
 note: candidate template ignored: could not match
  ’move_iterator’ against ’_List_iterator’
    operator-(const move_iterator<_IteratorL>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../
../../../include/c++/5.4.0/bits/stl_iterator.h:1145:5:
 note: candidate template ignored: could not match
 ’move_iterator’ against ’_List_iterator’
    operator-(const move_iterator<_Iterator>& __x,
```

Random Access Iterator

Example (Simple sort programm)

```
    operator-(const move_iterator<_Iterator>& __x,
    ^
In file included from sort_list.cpp:1:
In file included from /usr/bin/../lib/gcc/
x86_64-linux-gnu/5.4.0/../../../../include/c++/
5.4.0/algorithm:62:
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../..
/../../include/c++/5.4.0/bits/stl_algo.h:1878:18:
 error: invalid operands to binary expression
  (’std::_List_iterator<int>’ and ’std::_List_iterator<int>’)
      if (__last - __first > int(_S_threshold))
          ~~~~~~ ^ ~~~~~~~
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../
../../../include/c++/5.4.0/bits/stl_algo.h:1966:9:
 note: in instantiation of function template
```

Random Access Iterator

Example (Simple sort programm)

```
  specialization
      'std::__final_insertion_sort<std::_List_iterator<int>,
       __gnu_cxx::__ops::_Iter_less_iter>' requested here
          std::__final_insertion_sort(__first, __last, __comp)
               ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../
../../../include/c++/5.4.0/bits/stl_algo.h:4698:12:
 note: in instantiation of function template
 specialization 'std::__sort<std::_List_iterator<int>,
      __gnu_cxx::__ops::_Iter_less_iter>' requested here
      std::__sort(__first, __last, __gnu_cxx::__ops::__iter_le
           ^
sort_list.cpp:6:7: note: in instantiation of function template
        std::sort(std::begin(l), std::end(l) );
             ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
```

Random Access Iterator

Example (Simple sort programm)

```
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
      'const std::_Bit_iterator_base' for 1st argument
  operator-(const _Bit_iterator_base& __x, const _Bit_iterator
  ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
    operator-(const reverse_iterator<_Iterator>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
    operator-(const reverse_iterator<_IteratorL>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
    operator-(const move_iterator<_IteratorL>& __x,
    ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
    operator-(const move_iterator<_Iterator>& __x,
```

Random Access Iterator

Example (Simple sort programm)

```
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include
    operator-(const move_iterator<_Iterator>& __x,
    ^
2 errors generated.
```

Facts

- Errors handled in duck typing, show the call stack
- They may be interminable
- A mechanism to speed up compile time and reduce compiler error verbosity is needed
- An overloading mechanism is desiderable for templates

```
void f(int){ std::cout << "Into int f \n";}

void f(double){ std::cout << "Into double f \n";}

void f(std::string){std::cout << "Into std::string f \n";}

int main(){ f(5); }
```

Run time introspection

Example (Example Python)

```
class A(object):
    # Simply overrides the 'object.__str__' method.
    def __str__(self):
        return "I am a A"

class B(object):
    # A custom method for my custom objects that I want to ser
    def serialize(self):
        return "I am a B"

def serialize(obj):
    if hasattr(obj, "serialize"):#obj has attr. 'serialize'?
        if hasattr(obj.serialize, "__call__"): #is a method?'
            return obj.serialize()
    return str(obj) #else call _str_ method
```

Run time introspection

Example (Example Python)

```
def main():
    a = A()
    b = B()
    print(serialize(a)) # ’I am A’
    print(serialize(b)) # ’I am B’

if __name__ == "__main__":
    main()
```

Template Overloading

Example (Example C++)

```
template <class T>
struct hasSerialize
{
  typedef char yes[1];
  typedef yes no[2];

  template <typename U, U u> struct reallyHas;

  template <typename C>
  static yes&
  test(reallyHas<std::string (C::*)()const,&C::serialize>*)
  { }
  template <typename> static no& test(...) { }
  static const bool value = sizeof(test<T>(0))==sizeof(yes);
};
```

Template Overloading

Example (Example C++)

```
template <class T>
typename
std::enable_if<hasSerialize<T>::value, std::string>::type
 serialize(const T& obj)
{
    return obj.serialize();
}
template <class T>
typename
std::enable_if<!hasSerialize<T>::value, std::string>::type
 serialize(const T& obj)
{
    return obj.to_string();
}
```

Template Overloading

Example (Example C++)

```
struct A{
  std::string to_string() const{
  return "Hello from A";
  }
};
struct B{
  std::string serialize() const{
  return "Hello from B";
  }
};
int main(){
A a; B b;
std::cout << serialize(a) << '\n';  // prints Hello from A
std::cout << serialize(b) << '\n';  // prints Hello from B
}
```

Are we concise as a violinist?

Figure: How a C++ template developer sees itself

Are we playing with fire?

Figure: More realistic C++ template developer...

Ugly hacking is invariably clever. It is similar to playing the violin with ones feet. It is admirable that it can be done but its place is in the circus and not in the Conservatoire. A. Stepanov on Notes on Programming

What it is needed?

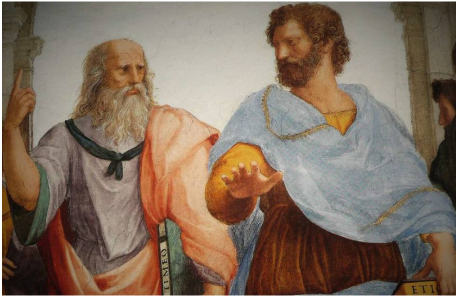

Figure: Plato and Aristoteles, the school of Athens

- Strong theoretical foundation
- Fit with current standard

Definitions

- Concepts are requirements on template arguments
- Concepts = constraints (sintactic) + axioms (semantics)
- Basically they establish "types" into templates
- A concept-lite is a compile-time predicate (that is, something that yields a Boolean value)

- The idea of concepts have been there since the 90s
- Debacle of concepts in c++x0 where 120 concepts were defined
- No strong theoretical background and no semantic (HasPlus, HasMinus ...)

Syntax and Semantics

- The behaviours invoked by different syntaxes, and for different types, are the semantics of those terms
- a + b is different for int and strings
- Semantic has to be similar and consistent for all the types
- Reduce cognitive burden

Abstract algebra

- Algebraic structures with a single binary operation: Magma, Quasigroup, Monoid, Semigroup, Group,
- Algebraic structures with a multiple operations: Ring, Field, Module, Vector space, Algebra over a field, Associative algebra, Lie algebra, Lattice, Boolean algebra

Regularity in C++ by A. Stepanov

- T a = b; assert(a==b);
- T a; a = b; ⟺ T a = b;
- T a = c; T b = c; a = d; assert(b==c);
- T a = c; T b = c; zap(a); assert(b==c && a!=b)

Concepts Lite proposal

Regularity	**Iterators**	**Functional**	**Types**
Comparable	Iterator	Function	Boolean
Ordered	`Forward_iterator`	Operation	
Copyable	`Bidirectional_iterator`	Predicate	
Movable	`Random_access_iterator`	Relation	
Regular			

Table: Concepts

Concepts Lite proposal

Operators	Language	Initialization	Other
Equal	Same	Destructible	Procedure
Less	Common	Constructible	Input_iterator
Logical_and	Derived	Assignable	Output_iterator
Logical_or	Convertible		
Logical_not	Signed_int		
Callable			

Table: Constraints

Axioms
Equivalence_relation
Strict_weak_order
Strict_total_order
Boolean_algebra

Keyword "concept" implicity constexpr(rvalue) and inline

Example

```
template<typename T, typename U>
concept bool
Equality_comparable()
{
  return requires (T a, U b) {
      {a == b} -> bool; // a can be compared to b using ==
                       // and returns something
                       // convertible to bool
     {a != b} -> bool;
     // Commutability
     {b == a} -> bool;
     {b != a} -> bool;
  };
};
```

Keyword ”requires”

Example

```
template<typename T>
 requires false
 class Bar{};

int main(){    Bar<int> f_bar;}

/usr/bin/g++-8 requires.cpp -fconcepts

requires.cpp: In function int main():
requires.cpp:143:9: error: template constraint failure
  Bar<int> f_bar;
         ^
requires.cpp:143:9: note:   constraints not satisfied
requires.cpp:143:9: note: false evaluated to false
```

Concepts declaration

Example

```
template<typename T>
 constexpr bool CopyConstructible()
 {    return  std::is_copy_constructible<T>::value;  }
template<typename T>
 concept bool CopyAssignable()
 {
   return requires(T a, T b){
   { a = b } -> void;    };
 }
 template<typename T>
 concept bool  Destructible()
 {
   return requires(T a){
   { a.˜T() } -> void;   };
 }
```

Concepts nesting

Example

```
template<typename It>
  concept bool Iterator()
  {
   return requires (It lvalue) {
       CopyConstructible<It>;
       CopyAssignable<It>;
       Destructible<It>;
      *lvalue;
      {++lvalue} -> It&
    };
  };
```

Concepts signature

Example

```
// C++0x signature
template<InputIterator InIter, class OutIter,
EquivalenceRelation < auto, InIter::value_type> Pred >
requires OutputIterator< OutIter, RvalueOf < InIter::value_typ
&&
HasAssign<InIter::value_type, InIter::reference>
&&
Constructible<InIter::value_type, InIter::reference>
&&
CopyConstructible<Pred> OutIter
unique_copy(InIter first, InIter last, OutIter result, Pred pr
```

Concepts signature

Example

```
// C++17 signature
template<typename InputIterator, typename T>
InputIterator find(InputIterator first, InputIterator last,
                   const T& value);

// C++20 signature?
template < InputIterator Iter, typename T >
requires EqualtiyComparable < Iter::value_type, T>
Iter find(Iter first, Iter last, const T& value);
```

Concepts declaration

Example

```
template<typename T, typename U>
concept bool
Equality_comparable()
{
  return requires (T a, U b) {
      {a == b} -> bool; // a can be compared to b using ==
                       // and returns something
                       // convertible to bool
     {a != b} -> bool;
     //Commutability
     {b == a} -> bool;
     {b != a} -> bool;
  };
};
```

Concepts Overloading

Example

```
template<Input_iterator I>
void advance(I& i, int n)

template<Bidirectional_iterator I>
void advance(I& i, int n)

template<Random_access_iterator I>
void advance(I& i, int n)
```

Why Concepts didnt make C++17?

You guys do know that GCC is an open source compiler. If you had wanted something different, you could have submitted a patch sometime in, say, the last 2 years.

Sadly, few such patches were received. I can count on one hand the number of people who have worked to make the implementation better. I cannot count the number of people who have lined up to throw stones.

And yes, this is how I am perceiving your comments.

Be sure in your next blog post to mention you have failed to investigate the problem more deeply than simply showing up to complain.

Andrew Sutton@reddit 2016

Standard commitee

Figure: C++17 meeting in Kona

https://github.com/mirazabal/meetup/tree/master/2018/june/concepts

Thank You!

Most relevant references

B. Stroustrup and A. Sutton (2012)
A Concept Design for the STL
N3351

B. Stroustrup (2017)
Concepts: The Future of Generic Programming
P0557r1

A. Sutton (2012)
Concepts in C++17
P0248

A. Stepanov and P. McJones (2009)
Elements of Programming
Printed book

- https://jguegant.github.io/blogs/tech/sfinae-introduction.html
- https://slides.com/manusanchez/concepts-lite
- and many others...