Expression Templates

Everyday Template Meta-Programming in Qt

Volker Krause
volker.krause@kdab.com
@VolkerKrause

Why?

You use TMP every day![*]

* Possibly without knowing

A quick journey into TMP

- Turing-complete programming language found inside C++ by accident
  - Conditionals using template specialization
  - Recursion

Incomprehensible Gibberish


```
template <int N> struct F {
    static int const value =
        F<N - 1>::value + F<N - 2>::value;
};
template <> struct F<1> {
  static int const value = 1;
};
template <> struct F<0> {
  static int const value = 0;
};
```

Without Syntactic Noise


```
f 0 = 0
f 1 = 1
f n = f(n – 1) + f(n - 2)
```

Functional Programming

- “Pure” functions: no side-effects
- Recursion instead of loops
- Strong type system
- Lazy evaluation
- Data structures modeled with terms

Data Structures

- Haskell list:
  `Node(1 Node(2 Node(3 Nil)))`
- TMP list:
  `node<1, node<2, node<3, void>>>`
- Purely syntactic terms
- Recursion and pattern matching

Compiler Errors


```
akonadi/itempayloadinternals_p.h: In instantiation of ‘Akonadi::Payload<T>::Payload(const T&) [with T = QObject]’:
akonadi/item.h:657:54:   required from ‘typename
boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type
Akonadi::Item::setPayloadImpl(const T&) [with T = QObject; typename
boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type = void]’
akonadi/item.h:635:3:   required from ‘void Akonadi::Item::setPayload(const T&) [with T = QObject]’
akonadi/tests/itemhydratest.cpp:114:21:   required from here
corelib/kernel/qobject.h:333:5: error: ‘QObject::QObject(const QObject&)’ is private
In file included from akonadi/item.h:28:0,
                 from akonadi/tests/itemhydratest.cpp:23:
akonadi/itempayloadinternals_p.h:288:40: error: within this context
In file included from QtCore/qmetatype.h:1:0,
                 from corelib/kernel/qvariant.h:48,
                 from corelib/tools/qlocale.h:45,
                 from corelib/io/qtextstream.h:48,
                 from QtCore/qtextstream.h:1,
                 from corelib/io/qdebug.h:50,
                 from kdebug.h:27,
                 from akonadi/entity.h:33,
                 from akonadi/item.h:26,
                 from akonadi/tests/itemhydratest.cpp:23:
corelib/kernel/qmetatype.h: In instantiation of ‘static int QMetaTypeId2<T>::qt_metatype_id() [with T = QObject]’:
corelib/kernel/qmetatype.h:230:44:   required from ‘int qMetaTypeId(T*) [with T = QObject]’
akonadi/itempayloadinternals_p.h:145:58:   required from ‘static int
Akonadi::Internal::PayloadTrait<T>::elementMetaTypeId() [with T = QObject]’
akonadi/item.h:658:3:   required from ‘typename
boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type
Akonadi::Item::setPayloadImpl(const T&) [with T = QObject; typename
boost::disable_if_c<Akonadi::Internal::PayloadTrait<T>::isPolymorphic, void>::type = void]’
akonadi/item.h:635:3:   required from ‘void Akonadi::Item::setPayload(const T&) [with T = QObject]’
akonadi/tests/itemhydratest.cpp:114:21:   required from here
corelib/kernel/qmetatype.h:169:80: error: ‘qt_metatype_id’ is not a member of ‘QMetaTypeId<QObject>’
corelib/kernel/qmetatype.h: In static member function ‘static int QMetaTypeId2<T>::qt_metatype_id() [with T =
QObject]’:
corelib/kernel/qmetatype.h:169:83: error: control reaches end of non-void function [-Werror=return-type]
```

It's just a backtrace!

- Program execution failure, not a compiler error
- We know how to read backtraces
  - It's unlikely there are fundamental issues in the libraries you are using
  - Follow the trace from the error back to your code

Again, why?


```
QString a, b, c;
...
QString d = a + b + c;
```

What's wrong with that?


```
QString a, b, c;
...
QString tmp = a + b;
QString d = tmp + c;
```

I can fix that!


```
QString a, b, c;
...
QString d = a;
d += b;
d += c;
```

C++11 fixed that!

- Rvalue references & move semantics reduce temporaries
- Doesn't solve the re-allocation issue

I can fix that too!


```
QString a, b, c;
...
QString d;
d.reserve(a.size() + b.size() +
          c.size());
d += a;
d += b;
d += c;
```

Laziness FTW!

- I'm too lazy to do that myself all the time, can't the compiler do it automatically?

- Can be implemented with lazy evaluation of the expression.

TMP to the rescue!

- Build syntax tree of entire expression
- Verify/transform entire expression (optional)
- Evaluate eventually to obtain result

→ Expression Templates

Magic!

QStringBuilder

Building the Structure

- Overload operator+() to just return the syntax structure
- No evaluation yet

```
Concat<QString, QString>

Concat<QString,
       Concat<const char[42],
       QString>>
```

Calculating the size

- Sum up the size of all sub-strings recursively
- Use template specialization to end recursion

Compute the result

- Implement cast operator to QString
- Create result QString with final size
- Iterate over all sub-strings and append them

Still, why?

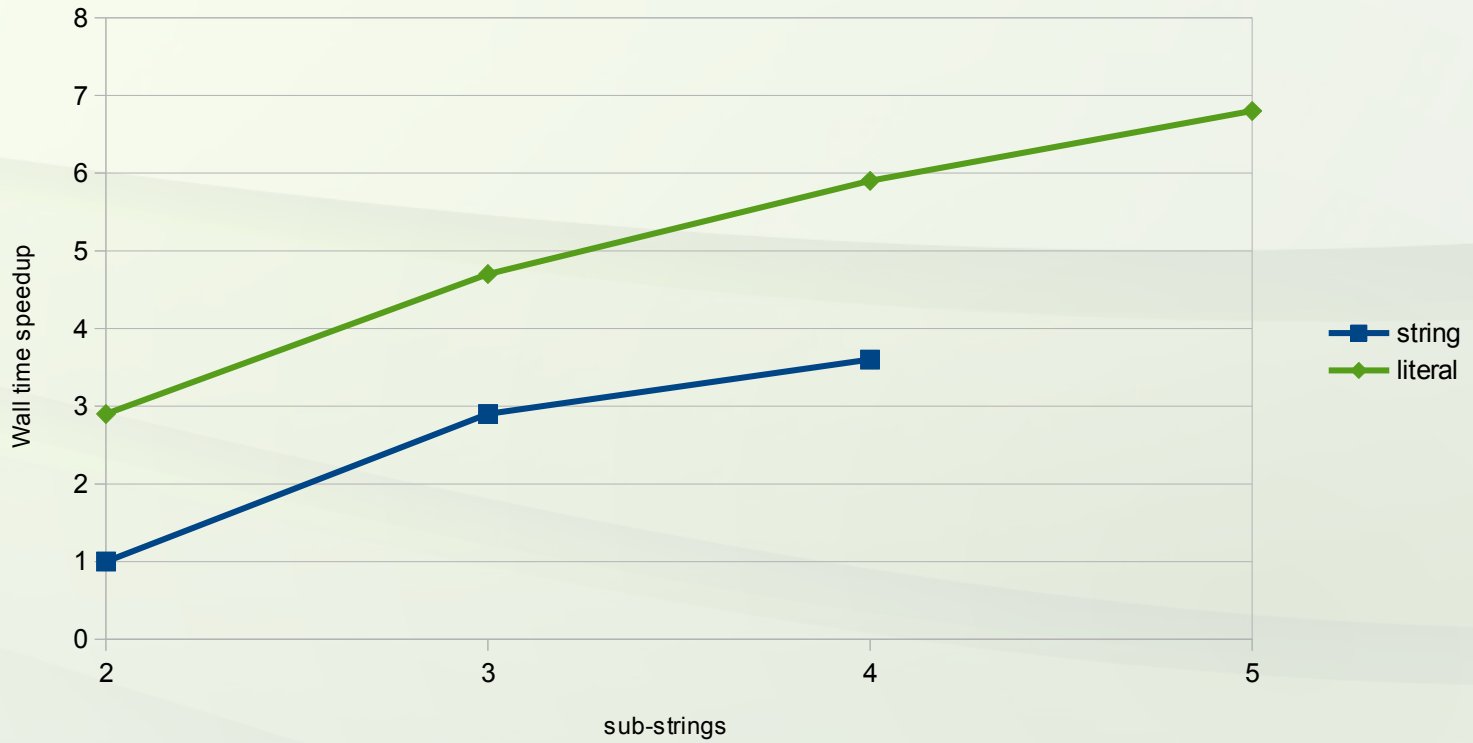
Is this really worth all the effort?

Faster!

Benchmark	Wall Time	Perf	Callgrind
2 literals	2.9x	2.9x	2.3x
2 strings	1.0x	1.0x	1.0x
2 string refs	3.3x	3.2x	3.5x
3 strings	2.9x	2.7x	2.9x
4 strings	3.6x	3.8x	3.6x
string + l1literal	2.5x	2.4x	2.1x
string + l1string	2.4x	2.4x	1.9x
3 literals	4.7x	4.4x	4.0x
4 literals	5.9x	5.4x	4.6x
5 literals	6.8x	7.7x	5.8x
4 chars	1.0x	1.0x	1.0x
char/string/char	1.1x	1.0x	1.0x

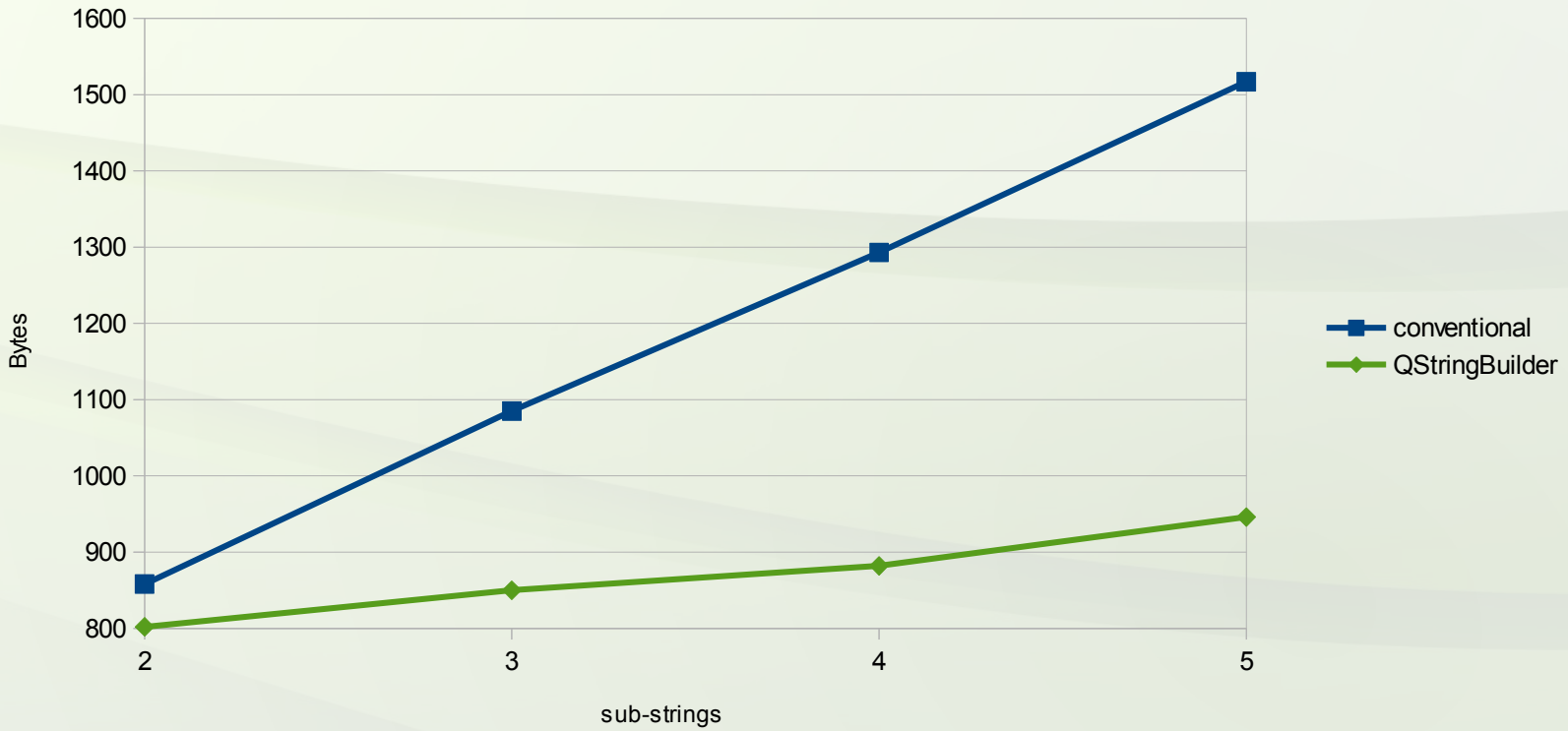
Really?

- Never slower than “normal” concatenation
- Speedup increases for more strings



But the code must be huge!

- Symbol names contain data structures
- It's all inline!



There must be a price...

Complexity

Using QStringBuilder

- `#define QT_USE_FAST_OPERATOR_PLUS`
- Almost source compatible
- `QString s = s1 % s2;`
- Also supports QByteArray

Implementation detail?

What could possibly go wrong...

Seeding Problem

- Operators for basic types can't be overloaded:

```
QString s = “Hello” + “World”;
```

- Need to have one custom type in there:

```
QLatin1Literal(“Hello”) + “World”;
```

Additional Conversion

- Works without QStringBuilder:

```
QString s = ...;
QVariant v = s + “literal”;
```

- Only one implicit conversion allowed:

```
v = QString(s + “literal”);
```

C++11 auto


```
QString s1(“Hello”);
auto s2 = s1 + “World”;
s1 = “Good Bye”;
QString s = s2;
```

Other Uses

Seen Andras Mantias lightning talk yesterday?

```
QSqlQuery q =
   select(Book.title, Author.name)
      .from(Book)
      .innerJoin(Author,
        Book.author == Author.id)
      .where(Book.price > 50.0
        && Book.price <= 100.0);
```

Conclusion

- TMP is “just” functional programming
- Read compiler errors as backtraces
- TMP is not just theory, you are using it already!

References

- David Vandervoorde, Nocolai M. Josuttis: “C++ Templates – The Complete Guide”, ISBN 0-201-73484-2, Addison-Wesly, 2003
- David Abrahams, Aleksey Gurtovoy: “C++ Template Metaprogramming”, ISBN 0-321-22725-5, Addison-Wesley, 2005
- Todd Veldhuizen, “Expression Templates”, C++ Report, Volume 7, 1995