CTSmarts

Compile time SMARTS expressions in C++20

Tim Vandermeersch

12th RDKit UGM, 20 September 2023, Mainz, Germany

github.com/timvdm/Kitimar

ABOUT ME

Tim Vandermeersch
Belgium
Software Developer (financial sector)
github.com/timvdm

Cheminformatics
- Pharmaceutical Sciences BSc
- OpenBabel
- Avogadro
- OpenSMARTS

OVERVIEW

1. Motivation
2. Generic Molecule API
3. CTSmarts
4. How Does It Work
5. Limitations
6. API
7. Compiler Explorer Demo
8. Future

Motivation

What does this function do?

```
bool is_???(auto &mol, const auto &atom)
{

    if (get_element(mol, atom) != 8)
        return false;
    if (get_heavy_degree(mol, atom) != 1)
        return false;

    auto nbr = null_atom(mol);
    for (auto bond : get_bonds(mol, atom)) {
        if (get_element(mol, get_nbr(mol, bond, atom)) == 16) {
            nbr = get_nbr(mol, bond, atom);
            break;
        }
    }
    if (nbr == null_atom(mol))
        return false;

    if (count_free_oxygens(mol, nbr) != 2)
        return false;

    for (auto bond : get_bonds(mol, nbr))
        if (get_element(mol, get_nbr(mol, bond, nbr)) == 7)
            return false;

    return true;
}
```

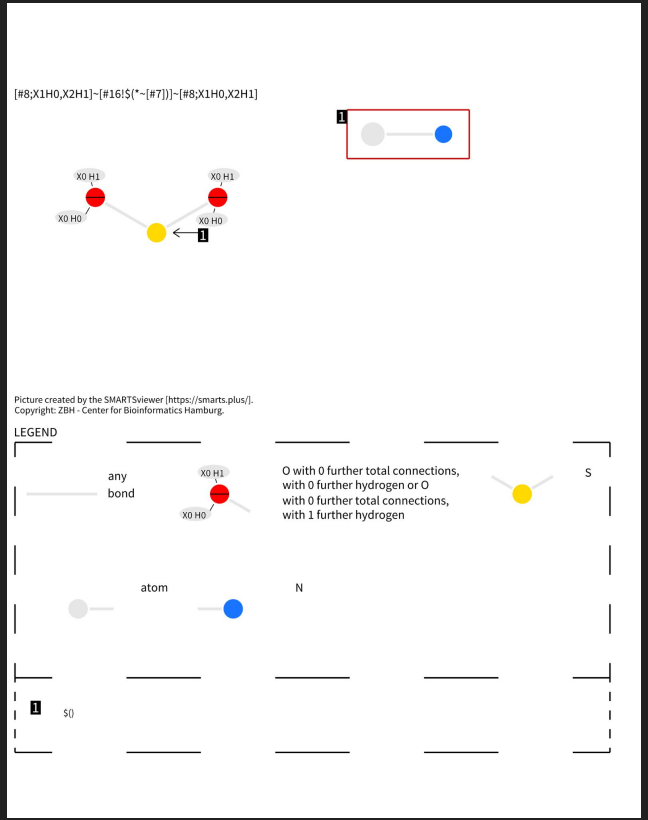
Is this easier to read?

- Function name
- Comments
- Part of algorithm
- Debug statements
- Not optimal
  - Slow iterators
  - 2 iterations over O's neighbors
  - 2 iterations over S's neighbors
- Reuses code
- Reuse this code?
  - e.g. match sulfone in mol

```
// Helper function for IsHBondAcceptor
static bool IsSulfoneOxygen(OBAtom* atm)
// Stefano Forli
//atom is connected to a sulfur that has a total
//of 2 attached free oxygens, and it's not a sulfonamide
//e.g. C-SO2-C
// Is this atom an oxygen in a sulfone(R1 - SO2 - R2) group ?
{
    if (atm->GetAtomicNum() != OBElements::Oxygen)
        return(false);
    if (atm->GetHvyDegree() != 1){
        //cerr << "sulfone> O valence is not 1\n";
        return(false);
    }

    OBAtom *nbr = nullptr;
    OBBond *bond1,*bond2;
    OBBondIterator i,j;

    // searching for attached sulfur
    for (bond1 = atm->BeginBond(i); bond1; bond1 = atm->NextBond(i))
        if ((bond1->GetNbrAtom(atm))->GetAtomicNum() == OBElements::Sulfur)
        { nbr = bond1->GetNbrAtom(atm);
            break; }
    if (!nbr){
        //cerr << "sulfone> atom null\n" ;
        return(false); }

    // check for sulfate
    //cerr << "sulfone> If we're here... " << atom->GetAtomicNum() <<"\n"
    //     << atom->GetAtomicNum() == OBElements::Sulfur << "\n";
    //cerr << "sulfone> number of free oxygens:" << atom->CountFreeOxygens() << "\n";
    if (nbr->CountFreeOxygens() != 2){
        //cerr << "sulfone> count of free oxygens not 2" << atom->CountFreeOxygens() << '\n';
        return(false); }

    // check for sulfonamide
    for (bond2 = nbr->BeginBond(j);bond2;bond2 = nbr->NextBond(j)){
        //cerr<<"NEIGH: " << (bond2->GetNbrAtom(atom))->GetAtomicNum()<<"\n";
        if ((bond2->GetNbrAtom(nbr))->GetAtomicNum() == OBElements::Nitrogen){
            //cerr << "sulfone> sulfonamide null\n" ;
            return(false);}}
    //cerr << "sulfone> none of the above\n";
    return(true); // true sulfone
}
```

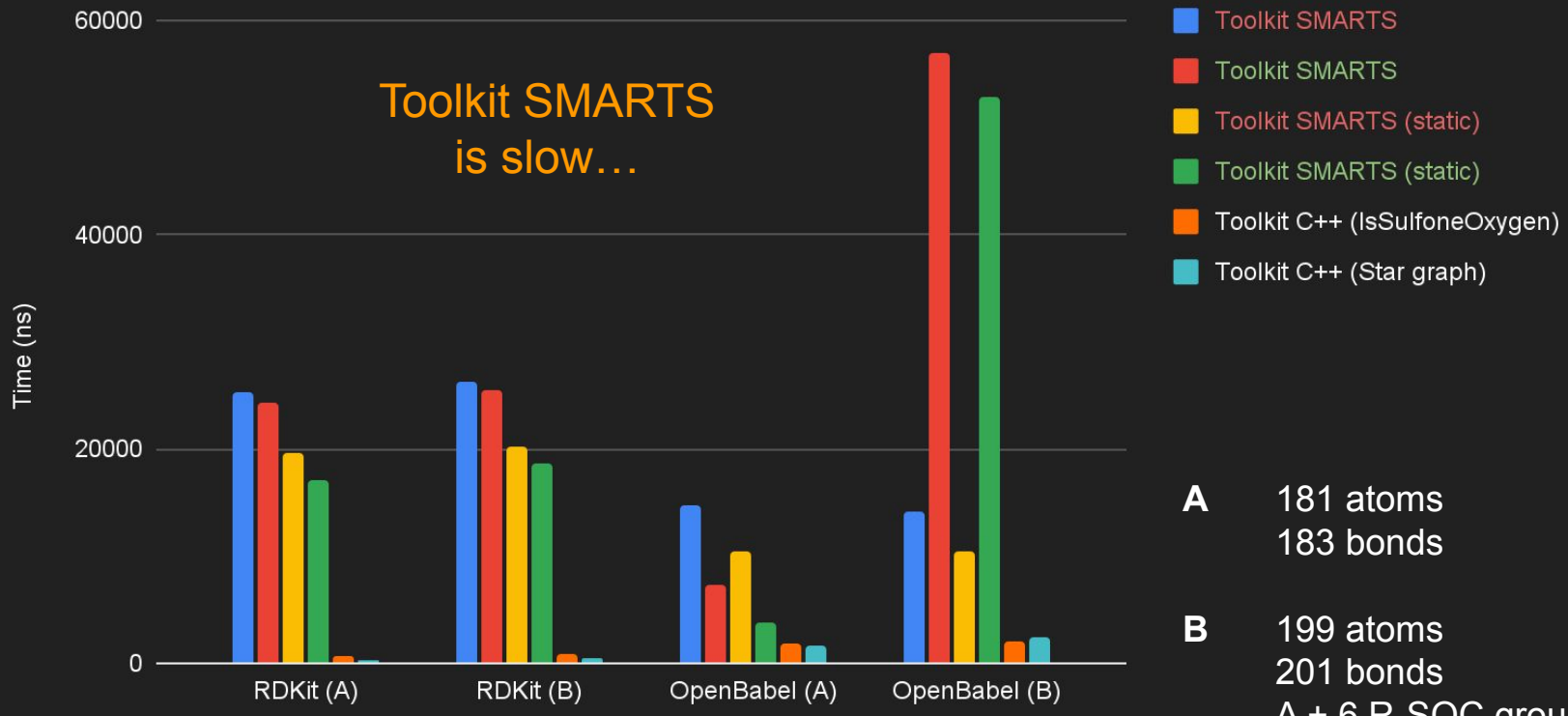
Why not use SMARTS? `[#8;X1H0,X2H1]~[#16!$(*~[#7])]~[#8;X1H0,X2H1]`

- SMARTS are for molecules, what regular expressions are for strings
- Toolkit agnostic
- Compact syntax
- Easy to read?
  - SMARTSview
  - SMARTSeditor
- Well documented
  - Daylight
  - OpenSMARTS (open standard)
  - Many implementations



Why not use SMARTS?

`[#8;X1H0,X2H1]~[#16!$(*~[#7])]~[#8;X1H0,X2H1]`
`[#16!$(*~[#7])](~[#8;X1H0,X2H1])~[#8;X1H0,X2H1]`



A 181 atoms
183 bonds

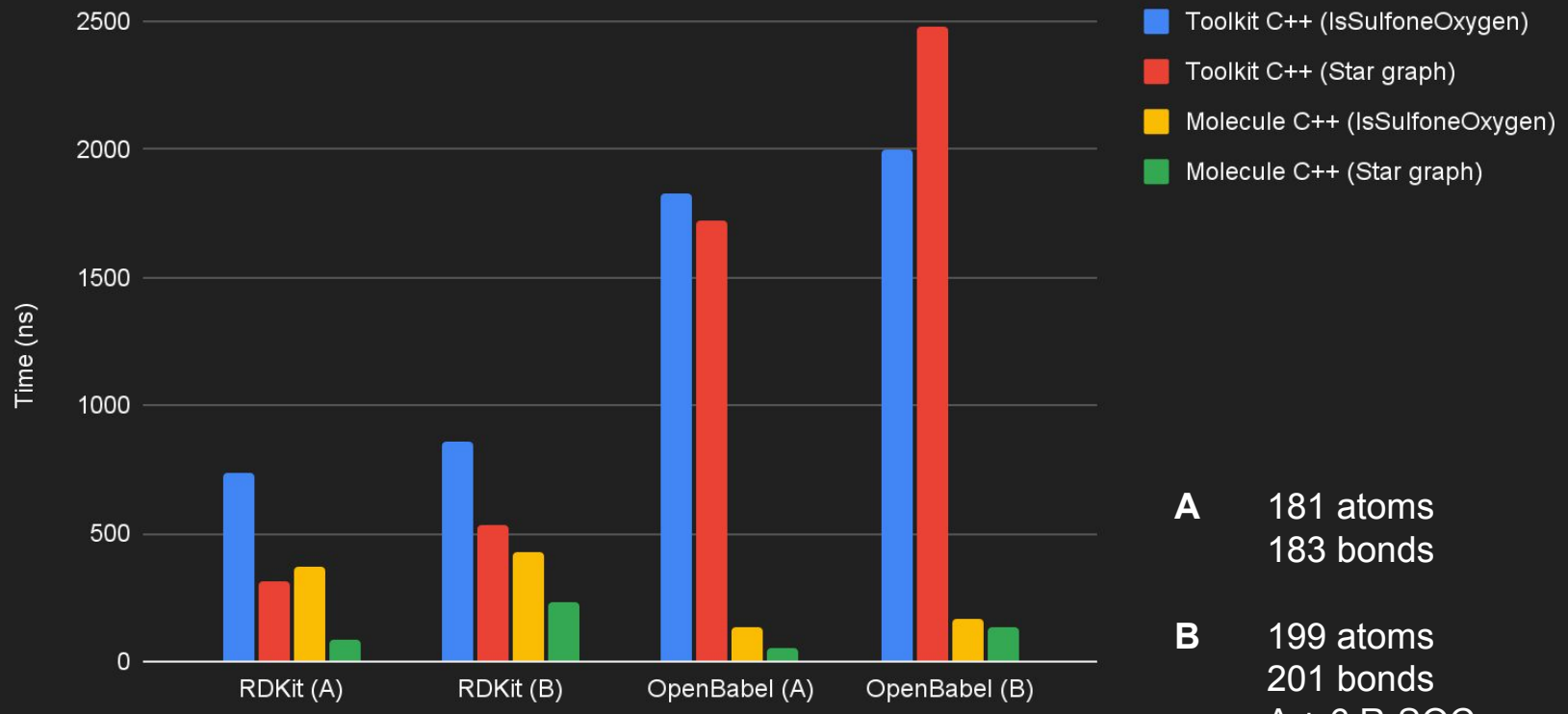
B 199 atoms
201 bonds
A + 6 R-SOC groups

Generic Molecule API

Generic Molecule API

- Wrapper around toolkit API
  - RDKit, OpenBabel, CTLayout (serialized molecules), …
- C++20 Molecule concept
  - Free functions
    - Inspired by Boost Graph Library (BGL)
      - Simplified
  - C++20 ranges over (adjacent) atoms and (incident) bonds
      - Range based for loops
- Part of Kitimar container project
- No overhead

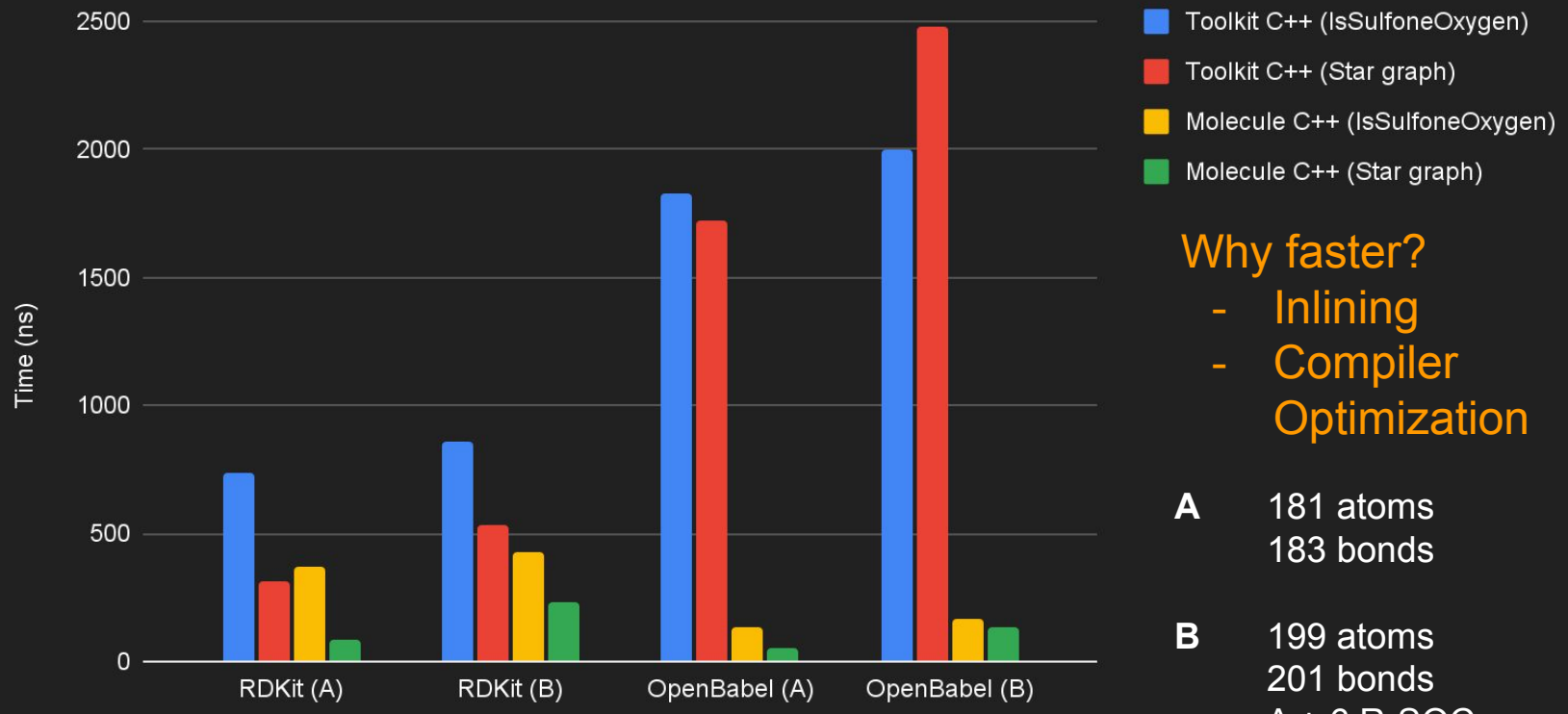
Generic Molecule API



A 181 atoms
183 bonds

B 199 atoms
201 bonds
A + 6 R-SOC groups

Generic Molecule API

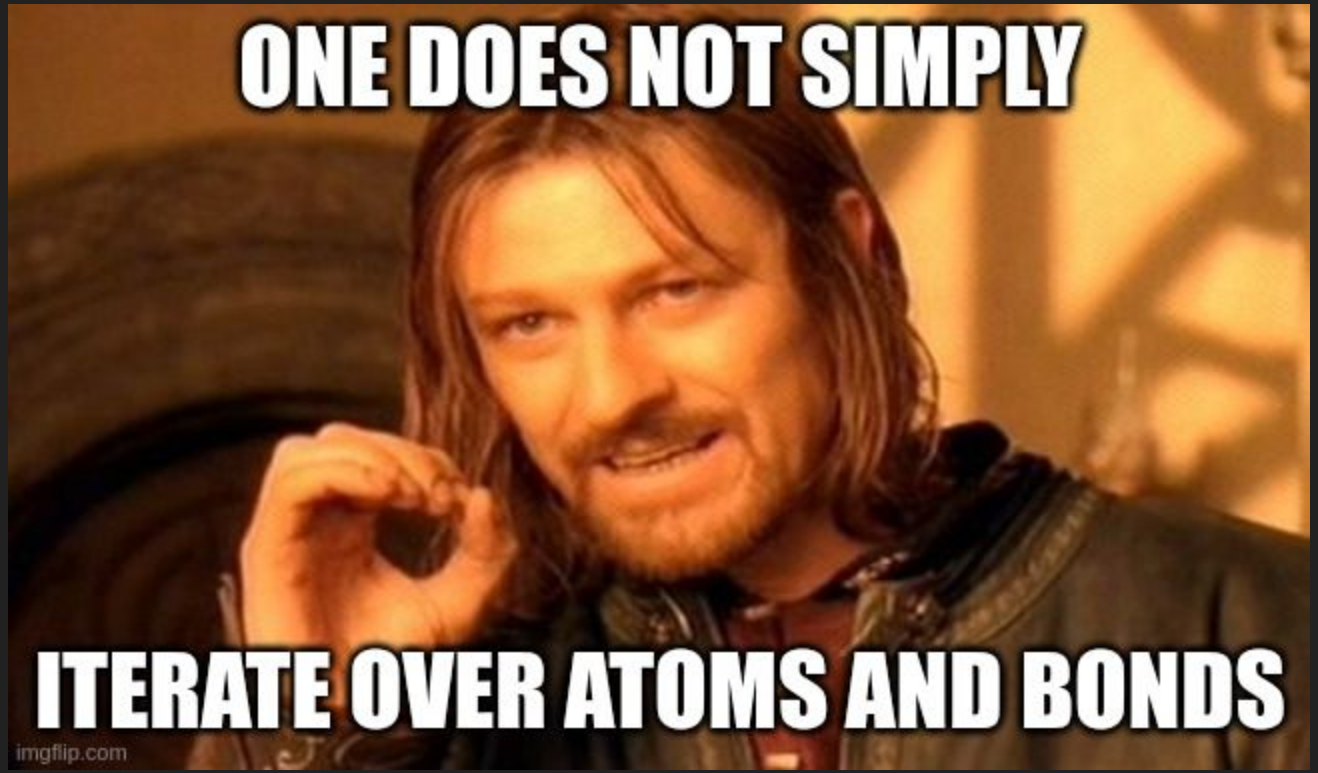


Why faster?

- Inlining
- Compiler Optimization

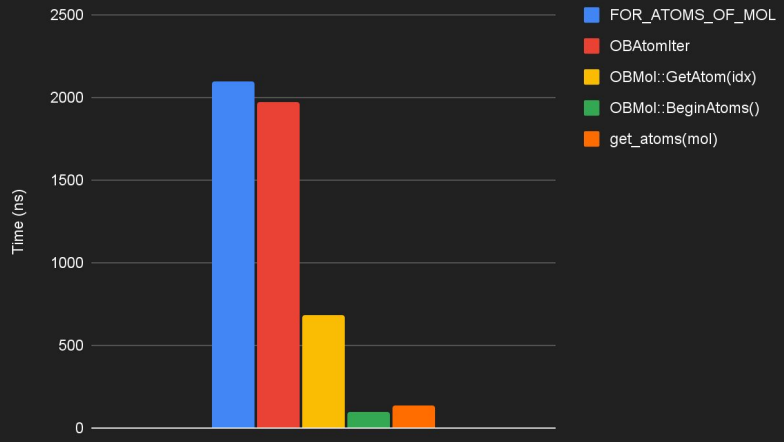
A 181 atoms
183 bonds

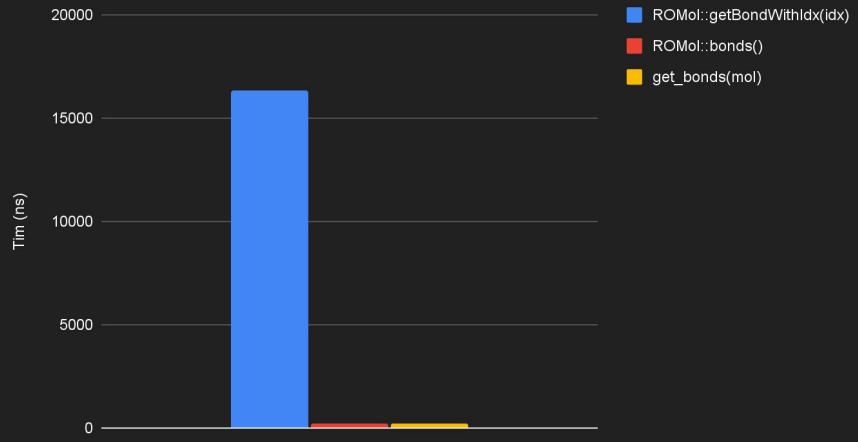
B 199 atoms
201 bonds
A + 6 R-SOC groups


ONE DOES NOT SIMPLY
ITERATE OVER ATOMS AND BONDS
imgflip.com

Looping over atoms and bonds

- https://www.rdkit.org/docs/GettingStartedInC%2B%2B.html#looping-over-atoms-and-bonds
  - `ROMol::getBondWithIdx(idx)`
- https://openbabel.org/api/2.2.0/classOpenBabel_1_1OBMol.shtml
  - `FOR_ATOMS_OF_MOL (atom, mol)`



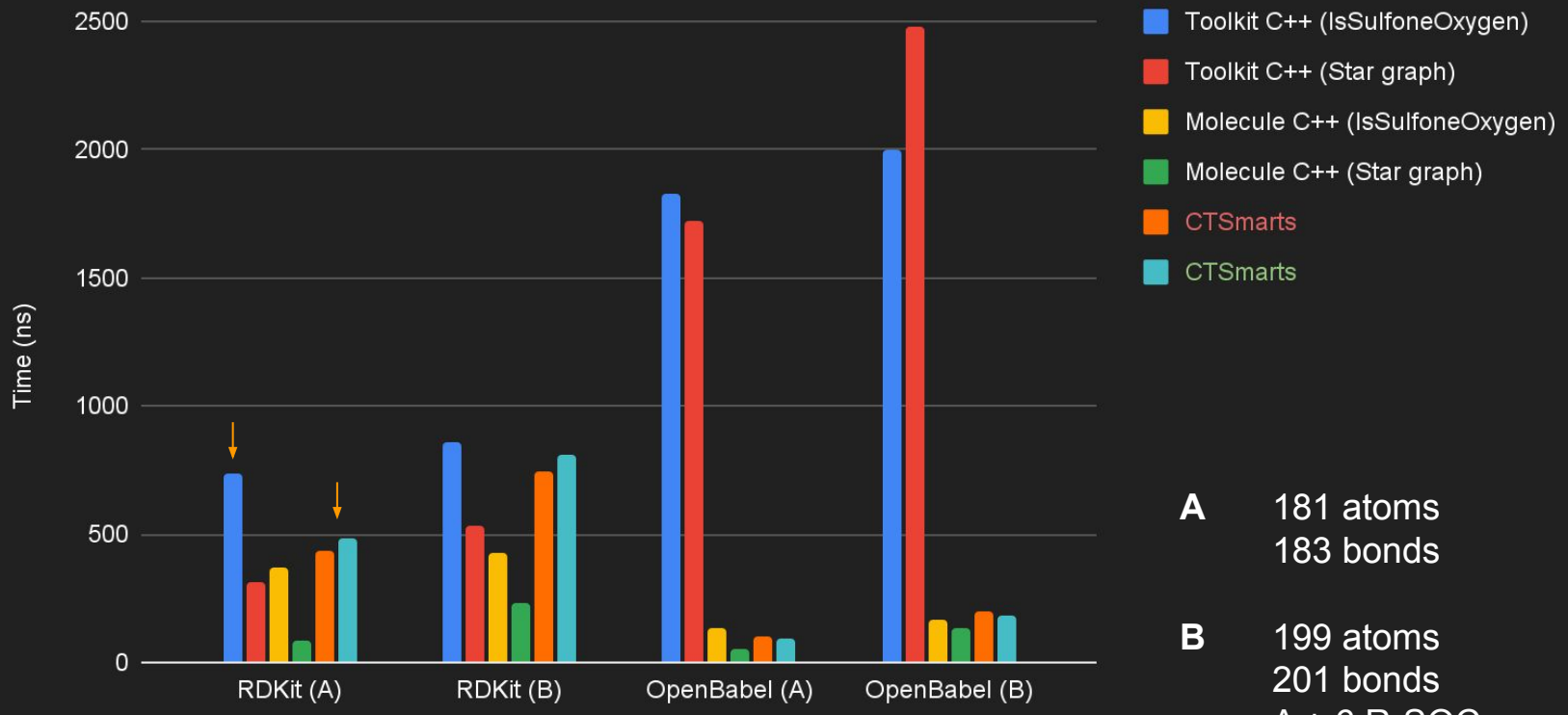


CTSmarts

```
bool contains_sulfone(const auto &mol)
{
    return ctse::match<"[#8;X1H0,X2H1]~[#16!$(*~[#7])]~[#8;X1H0,X2H1]">(mol);
}
```

CTSmarts

```
[#8;X1H0,X2H1]~[#16!$(*~[#7])]~[#8;X1H0,X2H1]
[#16!$(*~[#7])](~[#8;X1H0,X2H1])~[#8;X1H0,X2H1]
```

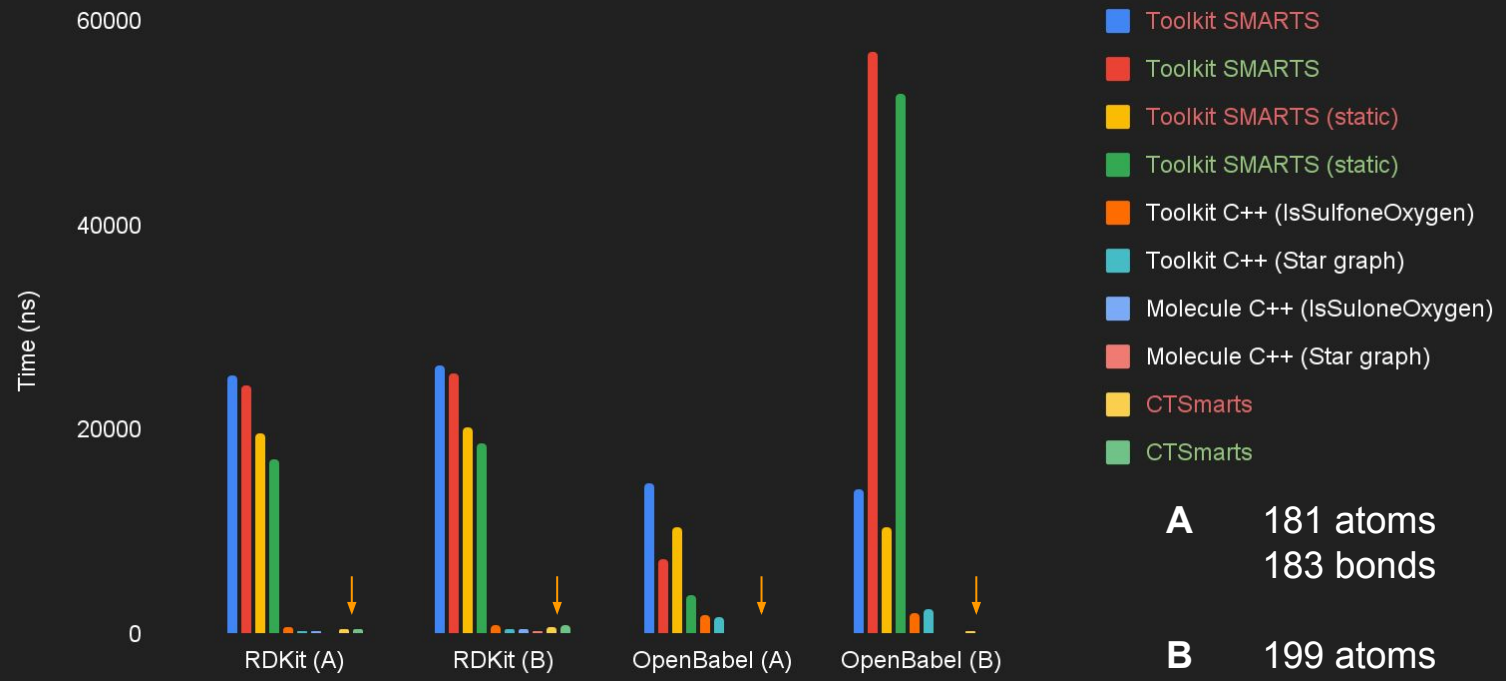


A 181 atoms
183 bonds

B 199 atoms
201 bonds
A + 6 R-SOC groups

CTSmarts

`[#8;X1H0,X2H1]~[#16!$(*~[#7])]~[#8;X1H0,X2H1]`
`[#16!$(*~[#7])](~[#8;X1H0,X2H1])~[#8;X1H0,X2H1]`



A 181 atoms
183 bonds

B 199 atoms
201 bonds
A + 6 R-SOC groups

CTSmarts

github.com/timvdm/Kitimar

- Compile-time SMARTS expressions in C++20
- Single header library
- Easy to use API
- Compilers
  - gcc >=11
  - clang >=16
- Compile-time (C++ constexpr)
  - Parsing
  - Optimize SMARTS expressions
  - Specialization: Atom, Bond, Chain *, Star *, …
- Extensible
- Fast! “Zero cost abstraction” *

Status

- First commit 13 May 2023
- Proof-of-concept 9 June 2023
- Release 0.1 17 September 2023
- Missing features
  - Stereochemistry, Components, …
- Known issues
  - Bad error reporting, …
- Validation
  - Substructure Query Collection (WIP)
- Early development
  - github.com/timvdm/Kitimar/issues

Previous Work

- Compile time regular expressions (CTRE)
  - github.com/hanickadot/compile-time-regular-expressions
  - Convert SMARTS string to C++ type AST
- NextMove Software
  - nextmovesoftware.com
  - Patsy / Arthor
  - SMARTS optimization
  - (byte)code generation
- Personal
  - OpenBabel MMFF94 atom typing (1175 LOC)
  - Helium
    - C++11 not good enough, a new toolkit with new bugs to discover :)
  - SmartsCompiler
    - Not novel, still no fast native C++ SMARTS expressions

Comparison

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

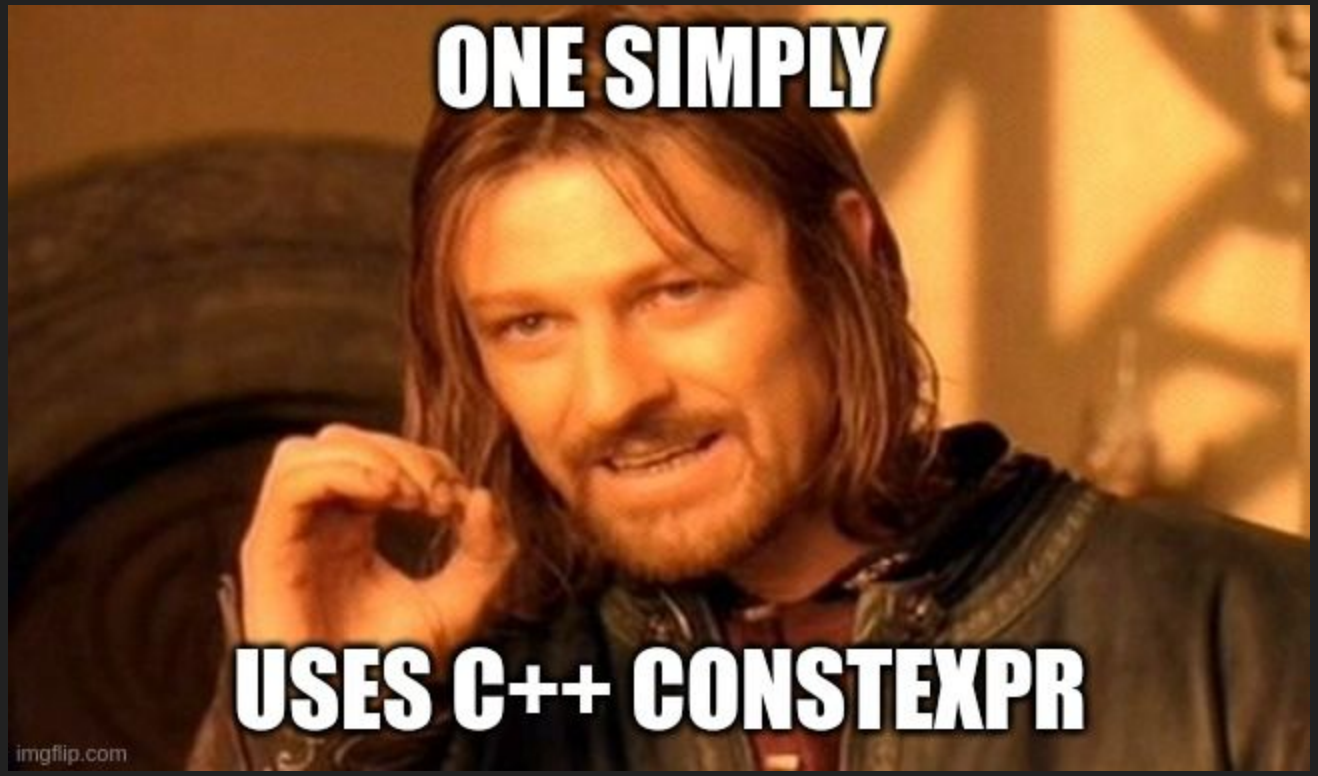
Native C++

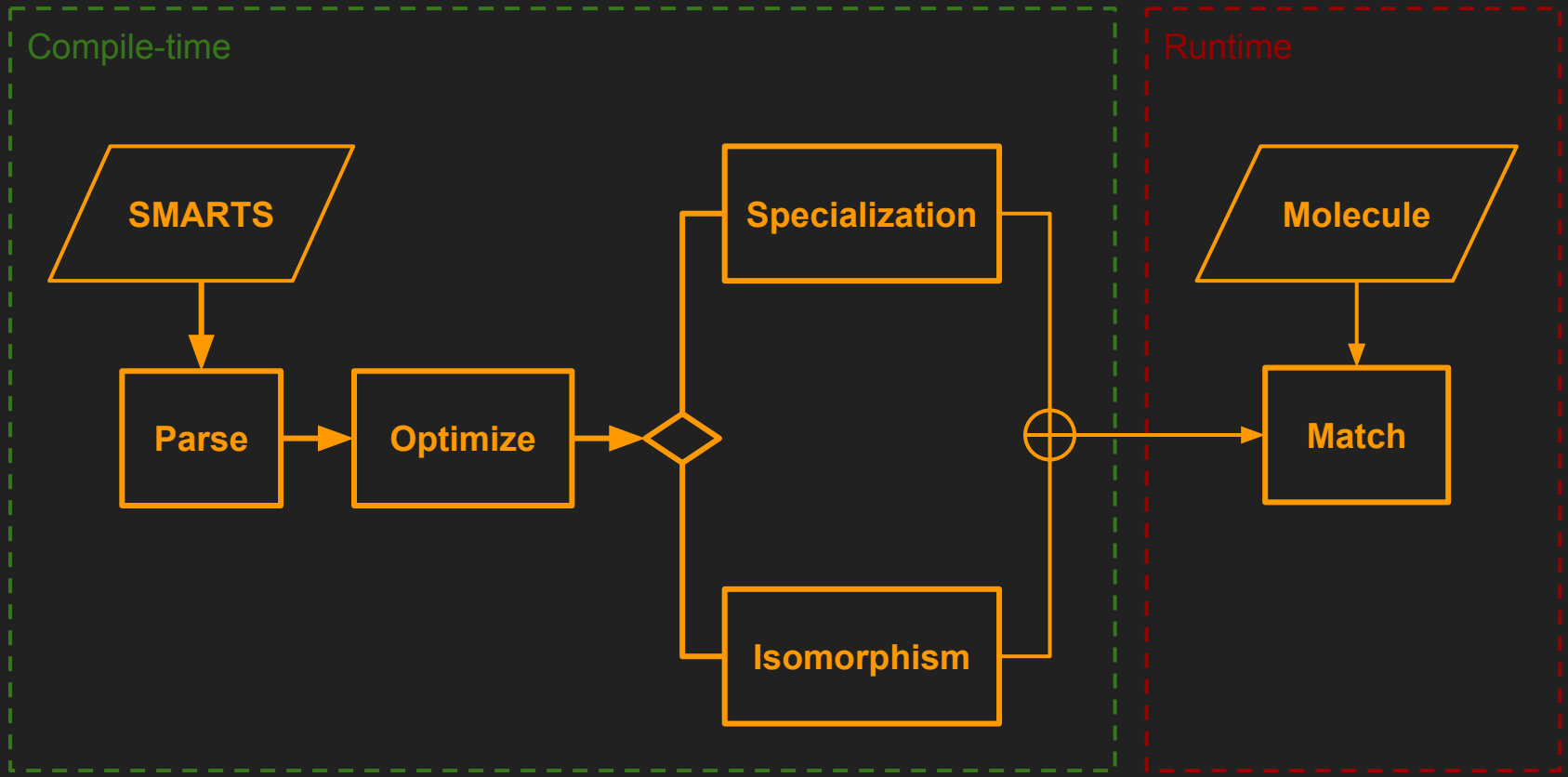
	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- A new way to use SMARTS
  - cfr. regex, STL algorithms
  - Beyond atom typing, filters, fingerprints, …
- Integrate SMARTS in algorithms

```
auto [found, C, O] = ctse::capture<"C=O">(mol);
if (found) {
    // Use atoms C and O
}

for (auto [C, O] : ctse::captures<"N[C:1]=[O:2]">(mol)) {
    // Use atoms C and O
}
```

How Does It Work


ONE SIMPLY
USES C++ CONSTEXPR
imgflip.com


Compile-time
Runtime
SMARTS
Parse
Optimize
Specialization
Isomorphism
Molecule
Match

Compile time computations

- C++11: `constexpr`
  - Very limited
- C++17: `constexpr if`
  - Return different types, alternative for template specialization
- C++20: transient constexpr allocations
  - That is, we can allocate during compile time - but only as long as the allocation is completely cleaned up by the end of the evaluation. [1]
- Constexpr data of unknown size
  - Use 2 passes: determine size, copy to std::array
  - Functional programming
    - return new values instead of modifying them
    - loop using recursion
- Value / Type based computations

Parsing

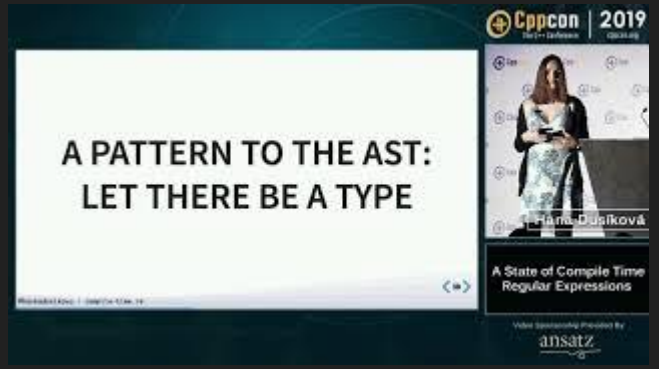
	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- Input SMARTS expression string

```
"[#6D3]"
```

- Output C++ Type based AST (`ctse::Smarts`)

```
template<int N> struct Degree {}
template<int N> struct Element {}
template<typename ...Expr> struct And {};
template<int Index, typename Expr> struct Atom {};

Atom< 0, And< Element<6>, Degree<3> > >;
```



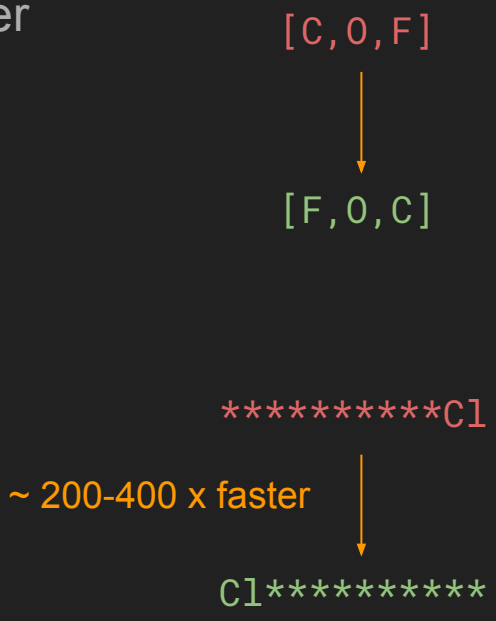
https://compile-time.re

Optimization

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- Input C++ Type based AST (`ctse::Smarts`)
- Output Bonds in optimized depth-first search order

- Use SMARTS primitive frequency
  - Reorder atom expressions
  - Start with least frequent atom
  - Continue with least frequent neighbor
- Additional optimizations
  - Add cycle membership primitives
  - Custom optimizers can be used



Matching

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- Input Bonds in optimized depth-first search order
- Output match, count, map(s)

- Simple isomorphism algorithm [1]
- No need for AST traversal (cfr. Toolkit) ⇔ cache misses
- No need for branching (cfr. Byte code) ⇔ branch prediction
- Every step (i.e. matching a bond) is a separate instance of a template function
  - Large binaries ⇔cache misses? generate byte-code? ⇒ measure!

Memory allocations

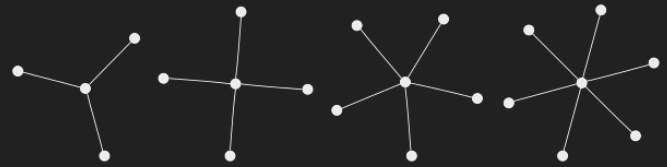
	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- None for specializations
- Current map
  - size known at compile time ⇒ std::array (no dynamic allocations)
- Mapped atoms index
  - size only known at runtime ⇒ std::vector<T> (dynamic allocations)
  - TODO: Use specialized bit vector (e.g. no dynamic allocations until # atoms > X)
- Unique count, maps, captures
  - Store hash of mapped atoms ⇒ std::vector<std::size_t> (dynamic allocations)
- maps and captures
  - return std::vector<Map>
  - TODO: validate that compiler can optimize this ⇒ if not, provide callback API
- Good, will improve

Specialization

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

- Specialize for specific graph classes
  - Single atom
  - Single bond
  - Chain *
  - Star *



- Specialize for search type
  - match, count, map(s), captures(s)
- Optimizes memory usage
  - No need for map(s) and mapped atoms index
  - Simplified isomorphism algorithm
  - Generated assembly same as toolkit C++

Specialization

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

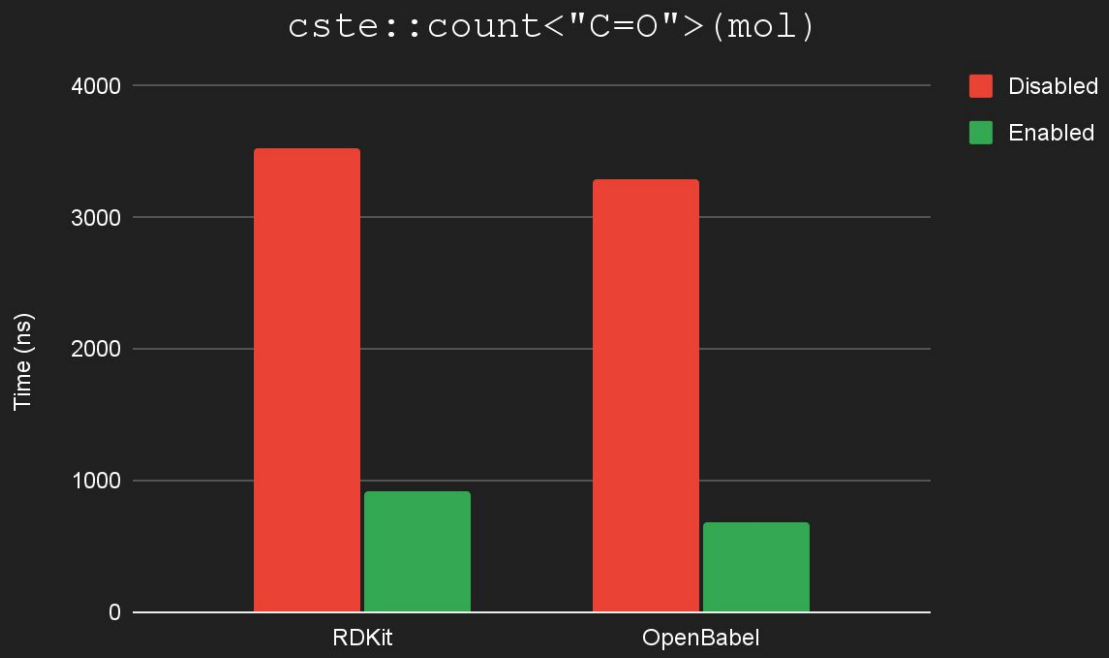
```
bool isCarbonDegree3_v1(auto &mol, auto atom)
{
    return get_element(mol, atom) == 6 && get_degree(mol, atom) == 3;
}


bool isCarbonDegree3_v2(auto &mol, auto atom)
{
    return ctse::match<"[#6D3]">(mol, atom);
}
```

Specialization

```
isCarbonDegree3_v1
 xor    eax,eax
 cmp    BYTE PTR [rdx+0x7],0x6
 jne    4010f6 <main+0x16>
 cmp    BYTE PTR [rdx+0xa],0x3
 sete   al
 movzx  eax,al
 ret

isCarbonDegree3_v2
 xor    eax,eax
 cmp    BYTE PTR [rdx+0x7],0x6
 jne    4010f6 <main+0x16>
 cmp    BYTE PTR [rdx+0xa],0x3
 sete   al
 movzx  eax,al
 ret
```

	Toolkit	CTSmarts	NextMove
Native C++	Yes	Yes	No
Parsing	Runtime	Compile-time	Pre compile-time
Optimization	N/A	Yes	Yes
Matching	AST traversal	C++	C++ / byte code
Memory Allocations	Many	None / Minimal	None / Minimal
Specialization	N/A	Yes	Yes

Specialization



Limitations

Dynamic Queries

- SMARTS must be known at compile time
- Not suitable for a molecular database search engine
- Possible solutions
  - Script: compile + run program
  - Just-in-time compilation?
  - Byte-code implementation?

“Zero cost abstraction” *

- Great, but…
- Hard to develop
- Compile time performance
  - Clang’s `-fexperimental-new-constant-interpreter` [1]
- Compile time memory usage
- Requires recent compiler
- Larger binaries

Compile time performance

VALIDATION SUITE

- Substructure Query Collection subset [1]
- 1900 SMARTS

PERFORMANCE

- clang 16.0 `4m49s` `~10GB`
- gcc 12.3 `6m02s` `~25GB`

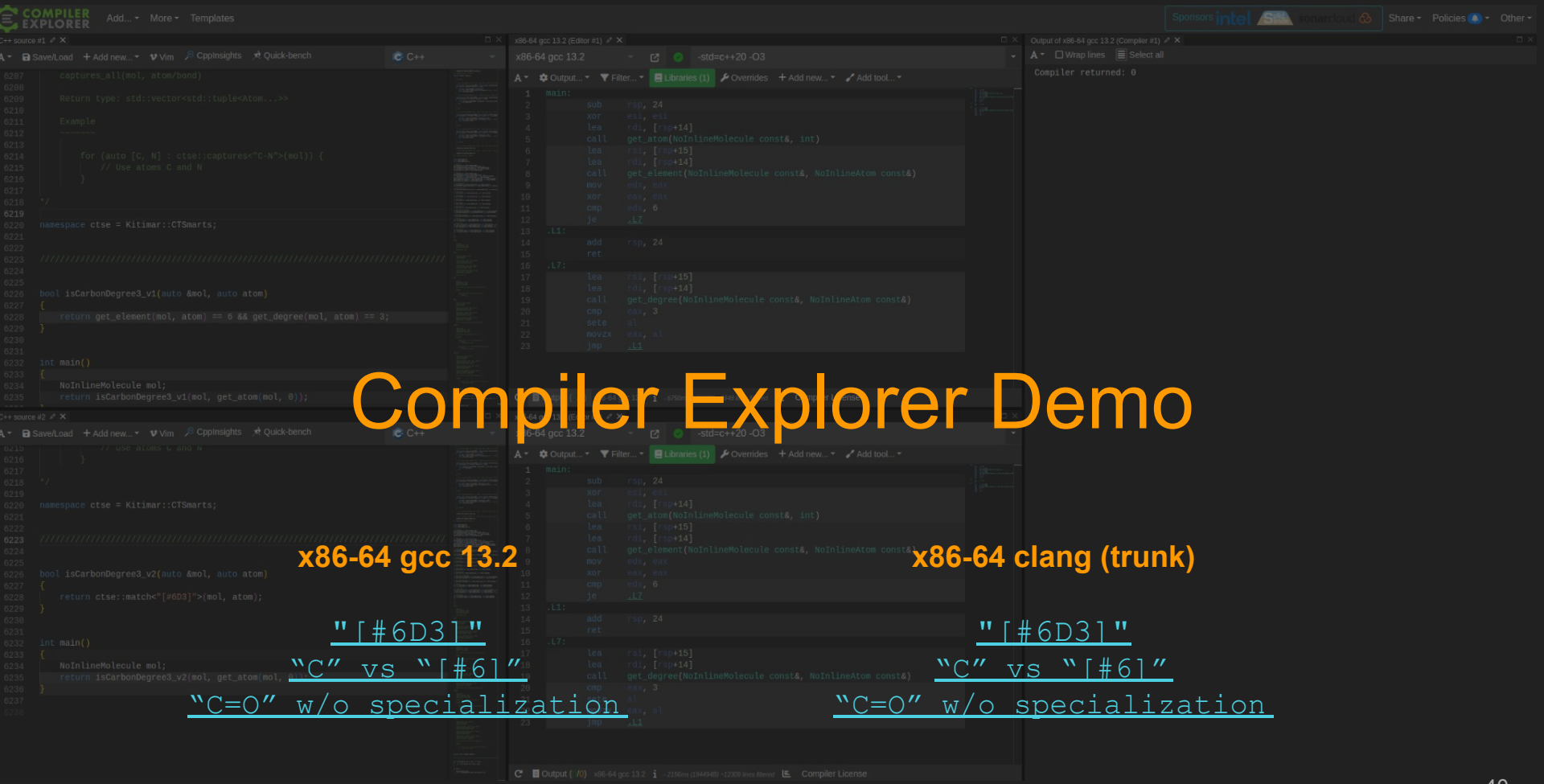
HARDWARE

- Intel Core i7-12700 (12 cores, 20 threads)
- 32GB DDR5 RAM, NVMe SSD storage

API

Match, Count, Map(s) & Capture(s)

```
// single
bool ctse::match<"SMARTS">(mol)
Map  ctse::map<"SMARTS">(mol)
auto [found, atoms...] = ctse::capture<"SMARTS">(mol)

// multiple results: _all / _unique
int  ctse::count_all<"SMARTS">(mol)
Maps ctse::maps_all<"SMARTS">(mol)
for (auto [atoms...] : ctse::captures_all<"SMARTS">(mol))

// seed: _atom / _bond (match first SMARTS atom/bond)
bool ctse::match_atom<"SMARTS">(mol, atom)

// overloading
int  ctse::count<"SMARTS">(mol, atom) // count_atom_unique
```



Compiler Explorer Demo

x86-64 gcc 13.2

"[#6D3]"

"C" vs "[#6]"

"C=0" w/o specialization

x86-64 clang (trunk)

"[#6D3]"

"C" vs "[#6]"

"C=0" w/o specialization

Future

- Release 1.0
  - Feature complete (OpenSMARTS compliant)
  - Stable API
  - Validated
- Performance
  - More specializations / optimizations
  - State machines (multi SMARTS matching)
- Features
  - SMARTS extensions
  - SMIRKS (`ctse::transform` algorithm)
  - …
- Project sustainability

Thank you

Questions?


ONE DOES NOT SIMPLY
KNOW ONE'S RUNTIME AT COMPILE TIME
imgflip.com

Extra

Star graph specialization

```
bool is_sulfone_sulfur(const auto &mol, const auto &atom)
{
    if (get_element(mol, atom) != 16)
        return false;

    auto numOxygens = 0;
    for (auto bond : get_bonds(mol, atom)) {
        auto nbr = get_nbr(mol, bond, atom);
        switch (get_element(mol, nbr)) {
            case 7:
                return false;
            case 8:
                switch (get_degree(mol, nbr)) {
                    case 1:
                        ++numOxygens;
                        break;
                    case 2:
                        if (!get_implicit_hydrogens(mol, nbr) && get_total_hydrogens(mol, nbr) == 1)
                            ++numOxygens;
                        break;
                    default:
                        break;
                }
                break;
            default:
                break;
        }
    }

    return numOxygens == 2;
}
```

Validation

ChEMBL 32 (2.3M molecules)

OpenBabel (first 250K molecules)

```
"B"     // OpenBabel matches aromatic “b” (1)
"[X0]" // X0 parsed as X1, bug in OpenBabel?
"[!H]" // implementation defined (match “[H]”, “c1ccccc1”, … ?)
"[Be,B,Al,Ti,Cr,Mn,Fe,Co,Ni,Cu,Pd,Ag,Sn,Pt,Au,Hg,Pb,Bi,As,Sb,Gd,Te]" // (1)
```

RDKit

```
"[!H]" // implementation defined (match “[H]”, “c1ccccc1”, … ?)
"[+H]"
"*!@*"
"[+,++,+++]" // “[+++]” not supported? (not standard)
"[-,--,---]" // “[---]” not supported? (not standard)
"*(!@*)(!@*)"
"[!#6]~*~*~[R]"
```

Validation

ChEMBL 32 (2.3M molecules)

RDKit (contd.)

```
"[R](-*(-*))~*~*~*~[a]"
"[#9,#17,#35,#53]~*(~*)~*"
"[F,Cl,Br,I]~*(~[!#1])~[!#1]"
"[!#1]~*(~[!#1])(~[!#1])~[!#1]"
"[!#1]~[!#6;!#1](~[!#1])~[!#1]"
"[$([cX3](:*):*),$([cX2+](:*):*)]"
"[!#6;!#1]~*(~[!#6;!#1])~[!#6;!#1]"
"[$([R]@1@[R]@[R]@[R]@[R]@[R]@[R]@[R]1),...]"
"[AR0]~[AR0]~[AR0]~[AR0]~[AR0]~[AR0]~[AR0]~[AR0]"
"[$([cX3](:*):*),$([cX2+](:*):*),$([CX3]=*),$([CX2+]=*)]"
"[Be,B,Al,Ti,Cr,Mn,Fe,Co,Ni,Cu,Pd,Ag,Sn,Pt,Au,Hg,Pb,Bi,As,Sb,Gd,Te]"
```

CTLayout

- Implements Molecule concept
- Data layout specified using C++ types
- Offsets/strides computed at compile time
- “Serialize” molecules
  - No need for deserialization
- Memory mapped files
- Fast!
- Experimental

Molecule adaptors

> Here's a radical proposal: no implicit hydrogens. All hydrogens are explicit. For argument's sake, I'll make the claim that implicit hydrogens were invented for VAX computers with actual core memory (those little ferrite gizmos threaded by tiny wires). In this day where any respectable computer has 4 to 128 GB of memory and 4 to 64 3 GHz CPUs, it's more work to keep track of hydrogens than it would be to just make them into ordinary atom objects. Those who want to only deal with heavy atoms could use a no-H iterator.

Craig James, eMolecules

Molecule adaptors

- Views on Molecules
- Store a minimal amount of data
- Range adaptors + Molecule API specializations
- Examples:
  - `Molecule::explicit_h(mol)`
  - `Molecule::implicit_h(mol)`
  - `ctse::subgraph<"SMARTS">(mol)`
  - `ctse::filter<"SMARTS">(mol)`