Clang matchers for verified usage of the C++ Standard Template Library

Gábor Horváth, Norbert Pataki

Presented by Christina Fosheim-Hoag, Sabrina Brogren, and Gabriel Matute

Agenda

- Motivation
- Approach
- Architecture
- Results
- Analysis

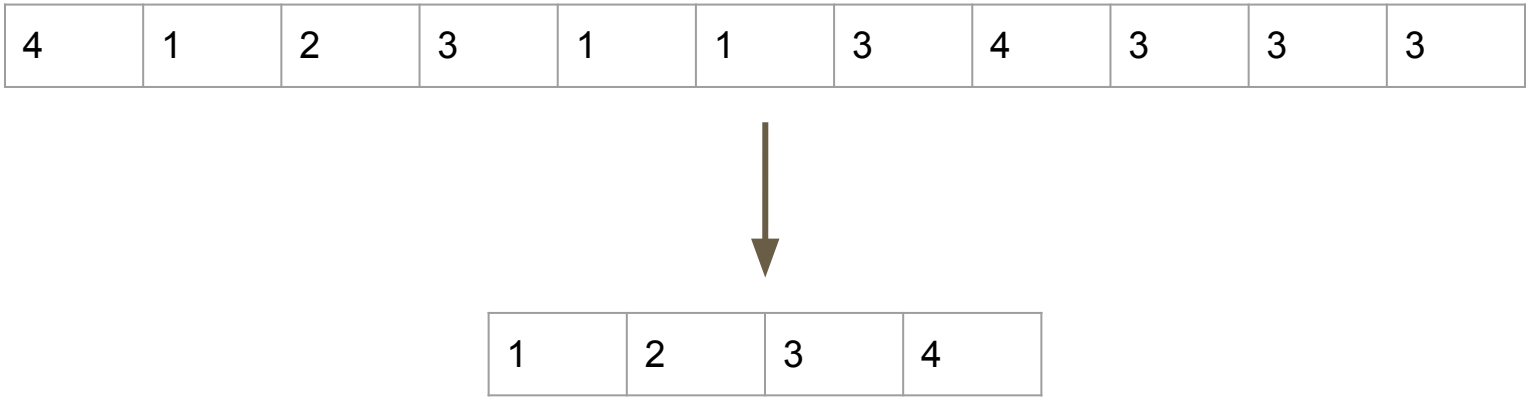
Motivation

- C++ Standard Template Library (STL) offers programmers code that is:
  - generic
  - extensible
  - maintainable
  - efficient

- But, does using the STL mean your code is error-free?

Motivation: Using `std::unique`

How many unique elements are in a vector?



Motivation: Using `std::unique`

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main(){
  vector<int> v{4, 1, 2, 3, 1, 1, 3, 4, 3, 3, 3};

  cout << "There are " << v.size() << " elements in the vector" << endl;

  unique(v.begin(), v.end());

  cout << "There are " << v.size() << " unique elements in the vector" << endl;
}
```

```
There are 11 elements in the vector
There are 11 unique elements in the vector
```

Motivation: Using `std::unique`

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main(){
  vector<int> v{4, 1, 2, 3, 1, 1, 3, 4, 3, 3, 3};

  cout << "There are " << v.size() << " elements in the vector" << endl;

  sort(v.begin(), v.end());
  unique(v.begin(), v.end());

  cout << "There are " << v.size() << " unique elements in the vector" << endl;
}
```

```
There are 11 elements in the vector
There are 11 unique elements in the vector
```

Motivation: Using `std::unique`

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main(){
  vector<int> v{4, 1, 2, 3, 1, 1, 3, 4, 3, 3, 3};

  cout << "There are " << v.size() << " elements in the vector" << endl;

  sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end());

  cout << "There are " << v.size() << " unique elements in the vector" << endl;
}
```

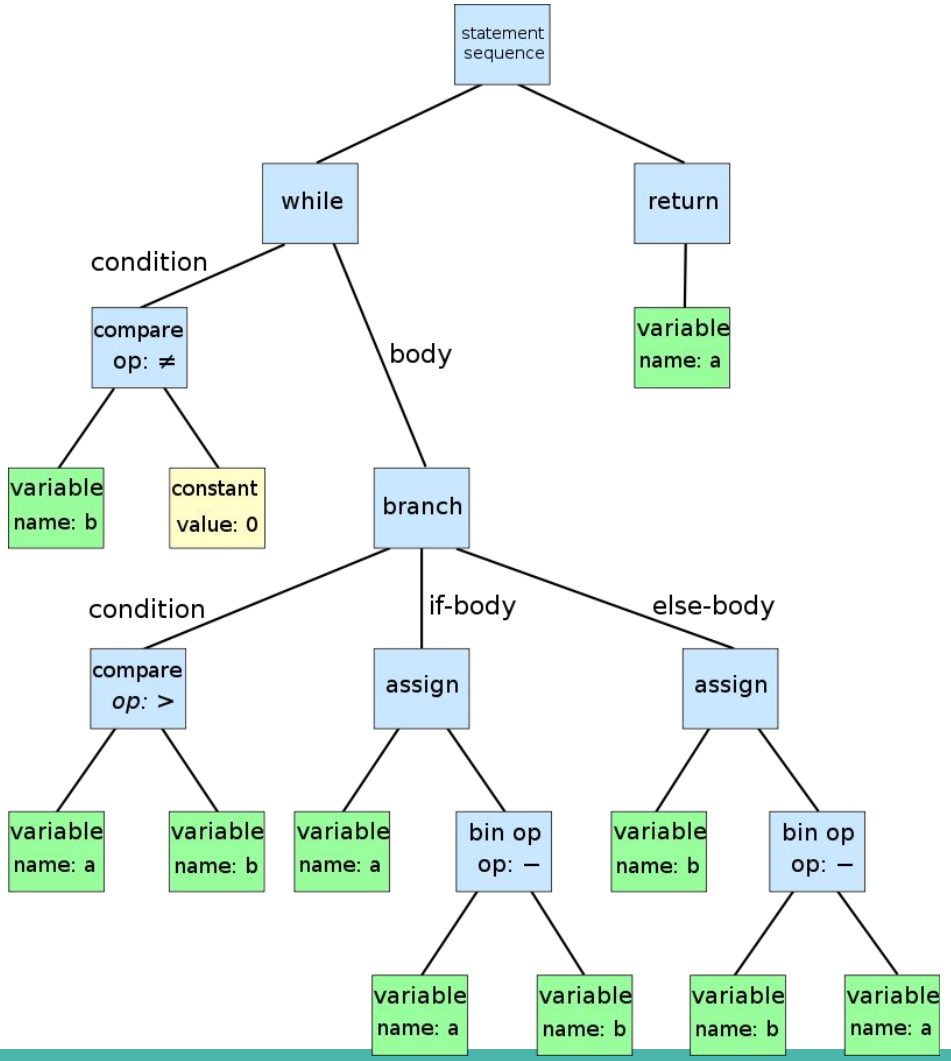
```
There are 11 elements in the vector
There are 4 unique elements in the vector
```

Approach

Match patterns on Abstract Syntax Trees (ASTs)

Approach: ASTs

```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```



Approach: Pattern Matching

Pattern

`sqrt(<expr1>)*sqrt(<expr2>)`

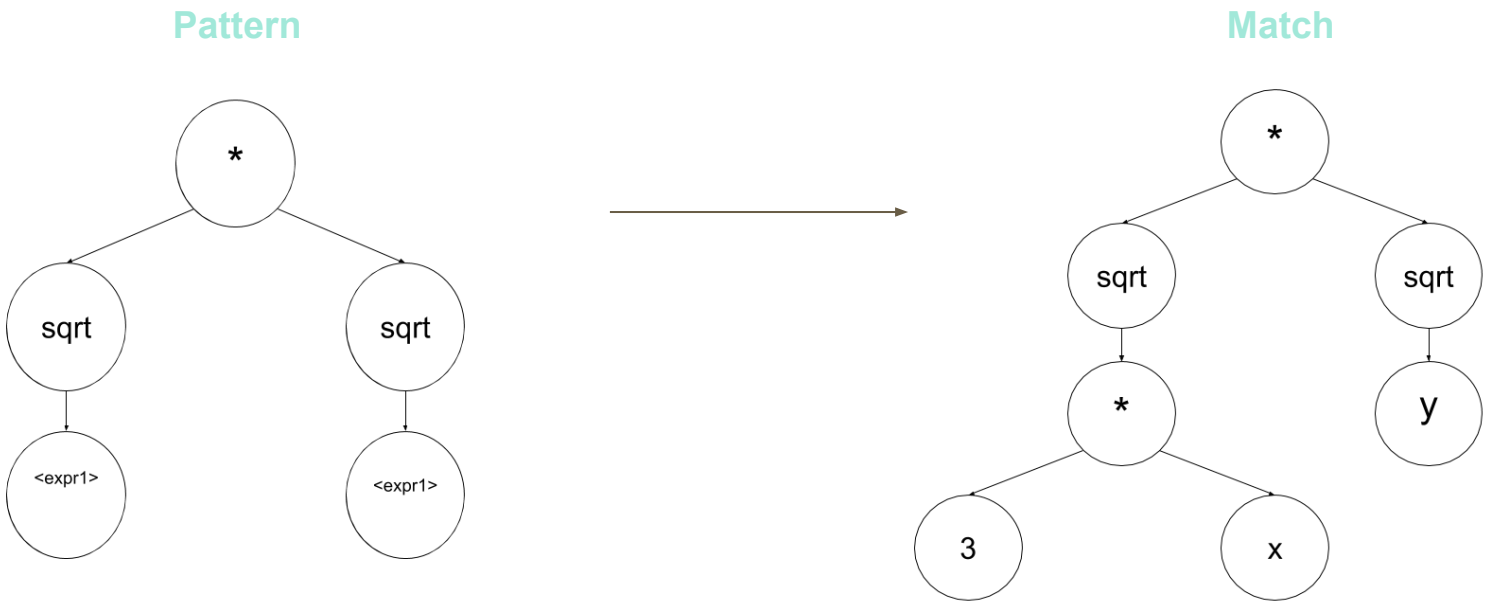
Matches

`sqrt(3*x)*sqrt(sqrt(z))`

`sqrt(4)*sqrt(9)`

`sqrt(func1(x*1))*sqrt(y)`

Approach: Pattern Matching with ASTs



Architecture: Predicates

Abstract interface to encapsulate static AST check

Architecture: Predicates

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Configure

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Activated?

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Matches

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Callback

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Excludes

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: ID

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Predicates: Bound ID

```
struct Predicate {
    Predicate() : _active(false) {}
    virtual bool matches(const MatchResult &result_) = 0;
    virtual void configure(std::string conf_) = 0;
    virtual std::string getErrorMsg() = 0;
    virtual std::string getID() = 0;
    virtual std::string getBoundID() = 0;
    virtual MatcherProxy getMatcher() = 0;
    virtual std::vector<MatchPosition>& getExcludes() {
        return _excludes;
    }
    virtual bool isActive() { return _active; }
    virtual void setActive(bool active_) { _active = active_; }
  protected:
    bool _active;
    std::vector<MatchPosition> _excludes;
};
```

Example: ASTMatcher

```
MatcherProxy StlFindCountPred::getMatcher() {
    return
        callExpr(
            callee(functionDecl(
                anyOf(hasName("std::find"),
                    hasName("std::count")))),
            hasArgument(0,
                hasDescendant(expr(hasType(typedefType(hasDecl(
                    matchesName(
                        "std::(multiset|set).*"
                        "::(const_iterator|iterator)"
                    ))))))).bind("id");
}
```

Example: ASTMatcher

```
MatcherProxy StlPolimoContPred::getMatcher()
{
    DeclarationMatcher container = unless(anything());
    for(const auto& e : gContainers)
        container = anyOf(recordDecl(hasName(e)), container);

    return
        implicitCastExpr(hasImplicitDestinationType(
            pointsTo(container))).bind("id");
}
```

Example: Callback

```
bool StlPolimoContPred::matches (const MatchFinder::MatchResult &result_) {

    const ImplicitCastExpr* cast =
        result_.Nodes.getDeclAs<ImplicitCastExpr>("id");

    return cast->getCastKind () == CK_DerivedToBase;
}
```

Results: Checks Implemented

- **Don't use `std::vector<bool>`.**
- Don't use containers of `std::auto_ptrs`.
- Don't use `std::find` or `std::count` on associative containers.
- Caution when using reverse iterator `base()`.
- Functors for the STL that require typedefs should be consistent.
- **Allocators must be stateless in C++98/C++03.**
- Functors for the STL should be stateless.

Results: Checks Implemented cont.

- STL functors should not use polymorphism.
- STL container empty check should use `empty()` instead of `size()`.
- Replace vector shrinking-to-fit idiom with `shrink_to_fit()`.
- **STL removing algorithms should be followed by an `erase()`.**
- STL copying algorithms should use iterator adaptors.
- STL containers should not be used polymorphically.
- **Should not assign integers to `std::string`.**

Results: It works

- "Some of the [issues] found by our tool are not detected by other static analysis tools available at the time of writing this article."

- "We found defects in the codebase of our industrial partner. However, they did not provide us with any data on the number of defects."

Analysis

Pros:

- Very useful for C++ novices
- Extensible framework, easy to add more checks
- Static checks can help modernize code and improve performance
- Many errors can be caught with simple AST matching

Cons:

- Compiler can only parse one translation unit at time
- Excludes list has a very bad UI