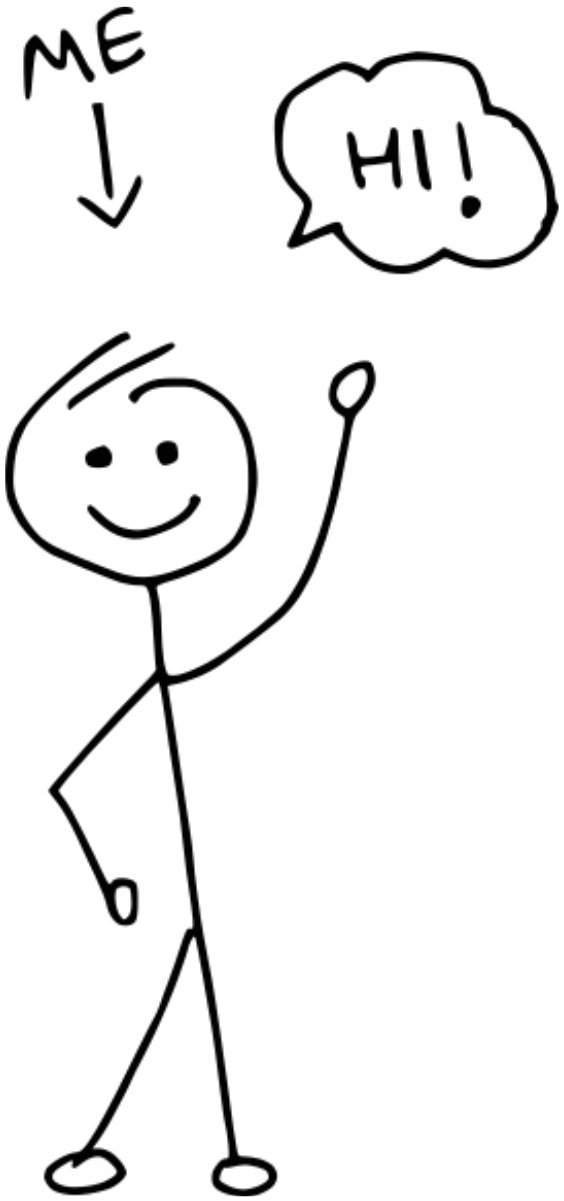
MLIR QUERY TOOL FOR EASIER EXPLORATION OF THE IR

EuroLLVM 2023

Student Talk

Devajith Valaparambil Sreeramaswamy



Devajith (Dave)

👨‍💻 Compiler Engineer **Intern** @ Huawei R&D UK

👨‍🎓 MSc Computing **Student** @ Cardiff University

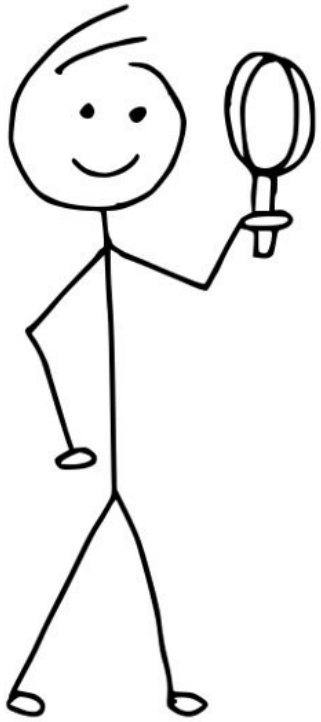
🌐 https://devajith.com

📧 devajithvs@gmail.com

Introduction

- Interactive query tool for MLIR
- REPL interface for querying various properties of MLIR code
- Can assist in debugging and testing MLIR
- Standalone tool

Basic Queries



basic-queries.mlir

```
module {
  func.func @basic_queries(%arg0: f32) -> f32 {
    %c2_i32 = arith.constant 2 : i32
    %0 = "hello.french"(%c2_i32) {bonjour = 1 : i32} : (i32) -> f32
    %1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32
    %2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32
    %3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32
    return %3 : f32
  }
}

```

Query: hasOpName

```
$ mlir-query basic-queries.mlir
mlir-query> m hasOpName("hello.japanese")

Match #1:

basic-queries.mlir:6:10: note: "root" binds here
%2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32

1 match.

mlir-query>
```

```
module {
  func.func @basic_queries(%arg0: f32) -> f32 {
    %c2_i32 = arith.constant 2 : i32
    %0 = "hello.french"(%c2_i32) {bonjour = 1 : i32} : (i32) -> f32
    %1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32
    %2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32
    %3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32
    return %3 : f32
  }
}

```

Query: hasOpAttrName

```
mlir-query> m hasOpAttrName("hola")

Match #1:

basic-queries.mlir:7:10: note: "root" binds here
%3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32

1 match.

mlir-query>
```

```
1 module {
2   func.func @basic_queries(%arg0: f32) -> f32 {
3     %c2_i32 = arith.constant 2 : i32
4     %0 = "hello.french"(%c2_i32) {bonjour = 1 : i32} : (i32) -> f32
5     %1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32
6     %2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32
7     %3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32
8     return %3 : f32
9   }
10 }
11
```

Query: isConstant

```
mlir-query> m isConstantOp()

Match #1:

basic-queries.mlir:3:15: note: "root" binds here
%c2_i32 = arith.constant 2 : i32

1 match.

mlir-query>
```

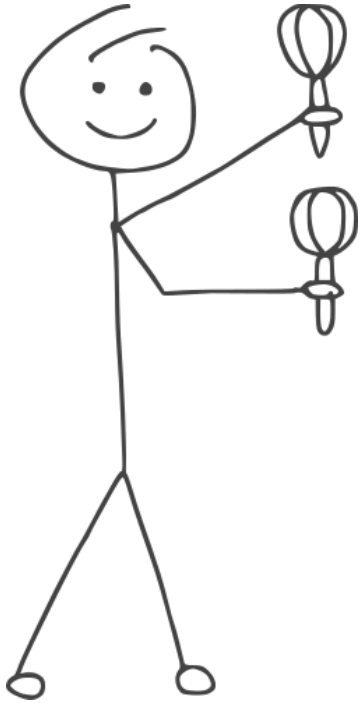
```
1 module {
2   func.func @basic_queries(%arg0: f32) -> f32 {
3     %c2_i32 = arith.constant 2 : i32
4     %0 = "hello.french"(%c2_i32) {bonjour = 1 : i32} : (i32) -> f32
5     %1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32
6     %2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32
7     %3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32
8     return %3 : f32
9   }
10 }
11
```

Query: anyOf

```
mlir-query> m anyOf(hasOpName("hello.english"), hasOpAttrName("konnichiwa"))

Match #1:

basic-queries.mlir:5:10: note: "root" binds here
%1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32

Match #2:

basic-queries.mlir:6:10: note: "root" binds here
%2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32

2 matches.

mlir-query>
```

```
module {
  func.func @basic_queries(%arg0: f32) -> f32 {
    %c2_i32 = arith.constant 2 : i32
    %0 = "hello.french"(%c2_i32) {bonjour = 1 : i32} : (i32) -> f32
    %1 = "hello.english"(%c2_i32) {hello = 1 : i32} : (i32) -> f32
    %2 = "hello.japanese"(%0, %1) {konnichiwa = 1 : i32} : (f32, f32) -> f32
    %3 = "hello.spanish"(%1, %2) {hola = 1 : i32} : (f32, f32) -> f32
    return %3 : f32
  }
}

```

More Advanced Queries



nested-queries.mlir

```
1 module {
2   func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3     %c1_i32 = arith.constant 1 : i32
4     "test.noop"() : () -> ()
5     %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6     %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7     %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8     %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9     %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11    %6 = "test.use_coo"(%5) : (i32) -> i32
12    return %6 : i32
13  }
14 }
```

Query: hasArgument

```
$ mlir-query nested-queries.mlir
mlir-query> m hasArgument(isConstantOp(), 2)

Match #1:

nested-queries.mlir:10:10: note: "root" binds here
%5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32

1 match.

mlir-query>
```

```
1 module {
2   func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3     %c1_i32 = arith.constant 1 : i32
4     "test.noop"() : () -> ()
5     %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6     %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7     %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8     %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9     %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11    %6 = "test.use_coo"(%5) : (i32) -> i32
12    return %6 : i32
13  }
14 }
```

Query: hasArgument

```
mlir-query> m hasArgument(hasArgument(isConstantOp(), 2), 0)

Match #1:

nested-queries.mlir:11:10: note: "root" binds here
%6 = "test.use_coo"(%5) : (i32) -> i32

1 match.

mlir-query>
```

```
1 module {
2   func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3     %c1_i32 = arith.constant 1 : i32
4     "test.noop"() : () -> ()
5     %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6     %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7     %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8     %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9     %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11    %6 = "test.use_coo"(%5) : (i32) -> i32
12    return %6 : i32
13  }
14 }
```

Query: uses

```
mlir-query> m uses(hasOpName("test.coo"))

Match #1:

nested-queries.mlir:11:10: note: "root" binds here
%6 = "test.use_coo"(%5) : (i32) -> i32

1 match.

mlir-query>
```

```
1 module {
2   func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3     %c1_i32 = arith.constant 1 : i32
4     "test.noop"() : () -> ()
5     %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6     %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7     %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8     %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9     %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11    %6 = "test.use_coo"(%5) : (i32) -> i32
12    return %6 : i32
13  }
14 }
```

Query: definedBy

```
mlir-query> m definedBy(hasOpName("test.coo"))

Match #1:

nested-queries.mlir:3:15: note: "root" binds here
%c1_i32 = arith.constant 1 : i32

Match #2:

nested-queries.mlir:5:10: note: "root" binds here
%0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32

Match #3:

nested-queries.mlir:9:10: note: "root" binds here
%4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32

3 matches.

mlir-query>
```

```
1  module {
2    func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3      %c1_i32 = arith.constant 1 : i32
4      "test.noop"() : () -> ()
5      %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6      %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7      %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8      %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9      %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10     %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11     %6 = "test.use_coo"(%5) : (i32) -> i32
12     return %6 : i32
13   }
14 }
```

Query: getDefinitions

```
mlir-query> m getDefinitions(hasOpName("test.coo"), 2)

Match #1:

nested-queries.mlir:6:12: note: "root" binds here
%1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)

Match #2:

nested-queries.mlir:8:10: note: "root" binds here
%3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32

2 matches.

mlir-query>
```

```
1  module {
2    func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3      %c1_i32 = arith.constant 1 : i32
4      "test.noop"() : () -> ()
5      %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6      %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7      %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8      %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9      %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10     %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11     %6 = "test.use_coo"(%5) : (i32) -> i32
12     return %6 : i32
13   }
14 }
```

Query: getAllDefinitions

```
mlir-query> m getAllDefinitions(hasOpName("test.coo"), 2)

Match #1:

nested-queries.mlir:3:15: note: "root" binds here
%c1_i32 = arith.constant 1 : i32

Match #2:

nested-queries.mlir:5:10: note: "root" binds here
%0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32

Match #3:

nested-queries.mlir:6:12: note: "root" binds here
%1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)

Match #4:

nested-queries.mlir:8:10: note: "root" binds here
%3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32

Match #5:

nested-queries.mlir:9:10: note: "root" binds here
%4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32

5 matches.

mlir-query>
```

```
module {
  func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
    %c1_i32 = arith.constant 1 : i32
    "test.noop"() : () -> ()
    %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
    %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
    %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
    %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
    %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
    %6 = "test.use_coo"(%5) : (i32) -> i32
    return %6 : i32
  }
}
```

Function extraction

```
mlir-query> m getAllDefinitions(hasOpName("test.use_coo"), 2).extract("test")


func.func @test(%arg0: i32, %arg1: i32, %arg2: i32, %arg3: i32) -> i32 {
  %c1_i32 = arith.constant 1 : i32
  %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
  %1 = "test.boo"(%arg2, %arg3) : (i32, i32) -> i32
  %2 = "test.coo"(%1, %0, %c1_i32) : (i32, i32, i32) -> i32
  return %2 : i32
}


mlir-query>
```

```
module {
  func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
    %c1_i32 = arith.constant 1 : i32
    "test.noop"() : () -> ()
    %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
    %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
    %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
    %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
    %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
    %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
    %6 = "test.use_coo"(%5) : (i32) -> i32
    return %6 : i32
  }
}
```

Function extraction

```
mlir-query> m anyOf(hasOpName("test.use_coo"),
getAllDefinitions(hasOpName("test.use_coo"), 2)).extract("test")


func.func @test(%arg0: i32, %arg1: i32, %arg2: i32, %arg3: i32) -> i32 {
  %c1_i32 = arith.constant 1 : i32
  %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
  %1 = "test.boo"(%arg2, %arg3) : (i32, i32) -> i32
  %2 = "test.coo"(%1, %0, %c1_i32) : (i32, i32, i32) -> i32
  %3 = "test.use_coo"(%2) : (i32) -> i32
  return %3 : i32
}


mlir-query>
```

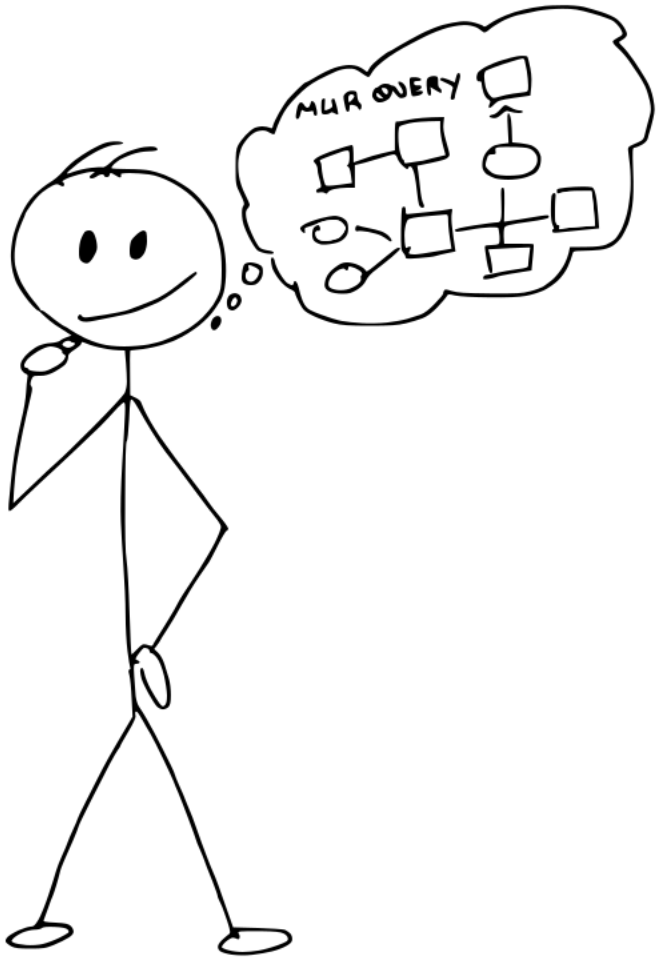
```
 1 module {
 2   func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
 3     %c1_i32 = arith.constant 1 : i32
 4     "test.noop"() : () -> ()
 5     %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
 6     %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
 7     %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
 8     %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
 9     %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10     %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11     %6 = "test.use_coo"(%5) : (i32) -> i32
12     return %6 : i32
13   }
14 }
```

Function extraction

```
mlir-query> m getAllDefinitions(uses(hasOpName("test.use_coo")), 3).extract("test")


func.func @test(%arg0: i32, %arg1: i32, %arg2: i32, %arg3: i32) -> i32 {
  %c1_i32 = arith.constant 1 : i32
  %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
  %1 = "test.boo"(%arg2, %arg3) : (i32, i32) -> i32
  %2 = "test.coo"(%1, %0, %c1_i32) : (i32, i32, i32) -> i32
  %3 = "test.use_coo"(%2) : (i32) -> i32
  return %3 : i32
}


mlir-query>
```

```
1  module {
2    func.func @foo(%arg0: i32, %arg1: i32, %arg2: i32) -> i32 {
3      %c1_i32 = arith.constant 1 : i32
4      "test.noop"() : () -> ()
5      %0 = "test.one_result"(%arg0, %arg1) : (i32, i32) -> i32
6      %1:2 = "test.many_results"(%0) : (i32) -> (i32, i32)
7      %2 = "test.unused_result"(%1#0, %1#1) : (i32, i32) -> i32
8      %3 = "test.foo"(%c1_i32, %1#1) : (i32, i32) -> i32
9      %4 = "test.boo"(%1#0, %3) : (i32, i32) -> i32
10     %5 = "test.coo"(%4, %0, %c1_i32) : (i32, i32, i32) -> i32
11     %6 = "test.use_coo"(%5) : (i32) -> i32
12     return %6 : i32
13   }
14 }
```

A Simplified Overview of the Implementation



- Matchers
- Parser
- Registry

MLIR Matchers

- Matchers are already available in MLIR
- Few matchers upstreamed as a part of the work on mlir-query
- Additional matchers still need to be upstreamed

Parser and Registry

- Parser for MLIR Query that parses query input
- Registry that maps existing MLIR matchers to mlir-queries.

What's Upcoming?

- Autocomplete and binding values
- More matchers!
- Clean-up, testing and optimization
- Patch upstream

Thanks

- To **Jacques Pienaar**
- To my **Colleagues at Huawei**
- To **you all**

QUESTIONS