

---

---

# IRDL: An IR Definition Language for SSA Compilers

— Gökay Gülsoy 270201072 —

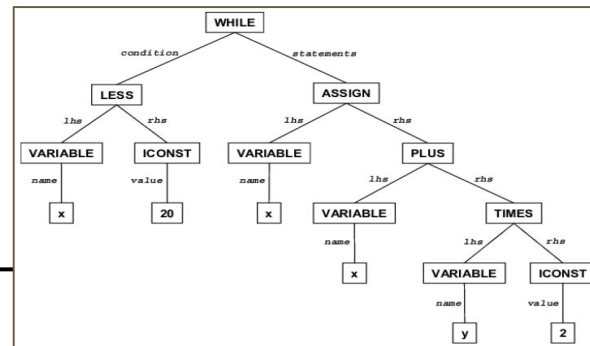
---

---

# IR Design for Compilers

- Developers generally use general-purpose programming languages to design IRs. As a result, IR implementations are verbose and manual modifications are expensive.
- While compilers relied historically on a few slowly evolving IRs, domain specific optimizations and specialized hardware motivate compilers to use and evolve many IRs.

```
; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc, i8** %argv) #0 {
entry:
  %retval = alloca i32, align 4
  %argv.addr = alloca i8**, align 8
  %argc.addr = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i8** %argv, i8*** %argv.addr, align 8
  store i32 %argc, i32* %argc.addr, align 4
  ret i32 0
}
```



# What is IRDL and Why it exists ?

- Domain-specific language to define IRs, and it facilitates the implementation of SSA-based IRs.
- It is capable of expressing 28 domain-specific IRs developed as a part of LLVM's MLIR project while only rarely falling back to IRDL's support for generic C++ extensions.
- Aim is to enable concise and explicit specification of IRs and provide foundations for developing effective tooling to automate compiler construction process.

# Need for IR design Languages

- LLVM, as a compiler infrastructure, has not only its user-facing LLVM IR but additionally uses various internal ones that are typically not visible to its users. Some of them are SelectionDAG, MachineInst, and MCInst.
- All of these IRs are deeply embedded into their respective compilers. Hence modifications require detailed compiler-specific knowledge, and even specialists are very hesitant to evolve existing IRs.
- While there exists approaches for generating parts of compiler (parsers, backends, code-generators, etc.), we lack a solution that streamlines the design of IRs themselves.

# General Properties of IRDL

- MLIR does not provide predefined set of operations but relies on the concept of extensibility, with few built-in constructs leaving most of the IR customizable.
- Operations, types, and attributes are grouped into dialects, similar to namespaces or modular libraries. Each dialect sits at a abstraction level. For example, Linalg dialect in MLIR models linear algebra operations on either tensor or buffer operands.

```
func @conorm(%p: !cmath.complex<!f32>, %q: !cmath.complex<!f32>) -> !f32
  %norm_p = cmath.norm(%p) : !f32
  %norm_q = cmath.norm(%q) : !f32
  %pq = std.mulf %norm_p, % norm_q : !f32
  return %pq : !f32
```

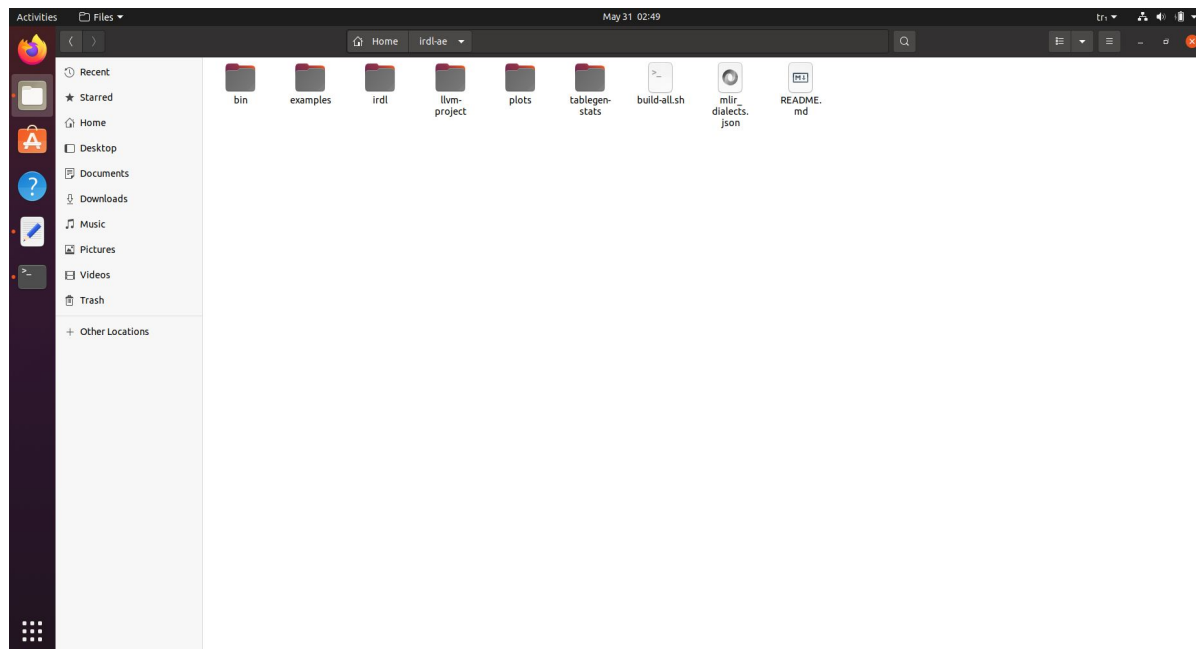
# Self-Contained IRDL specification

```
Dialect cmath {  
  Alias !FloatType = !AnyOf<!f32,!f64>  
  
  Type complex {  
    Parameters (elementType: !FloatType)  
  
    Summary “A complex number”  
  }  
  
  Operation mul {  
    ConstraintVar (!T: !complex<FloatType>)  
    Operands (lhs: !T, rhs: !T)  
    Results (res: !T)  
  
    Format “$lhs, $rhs: $T.elementType”  
    Summary “Multiply two complex numbers”  
  }
```

```
Operation norm {  
  ConstraintVar (!T: !FloatType)  
  Operands(c: !complex<!T>)  
  Results (res: !T)  
  
  Format “$c: $T”  
  Summary “norm of complex number”  
}  
}
```

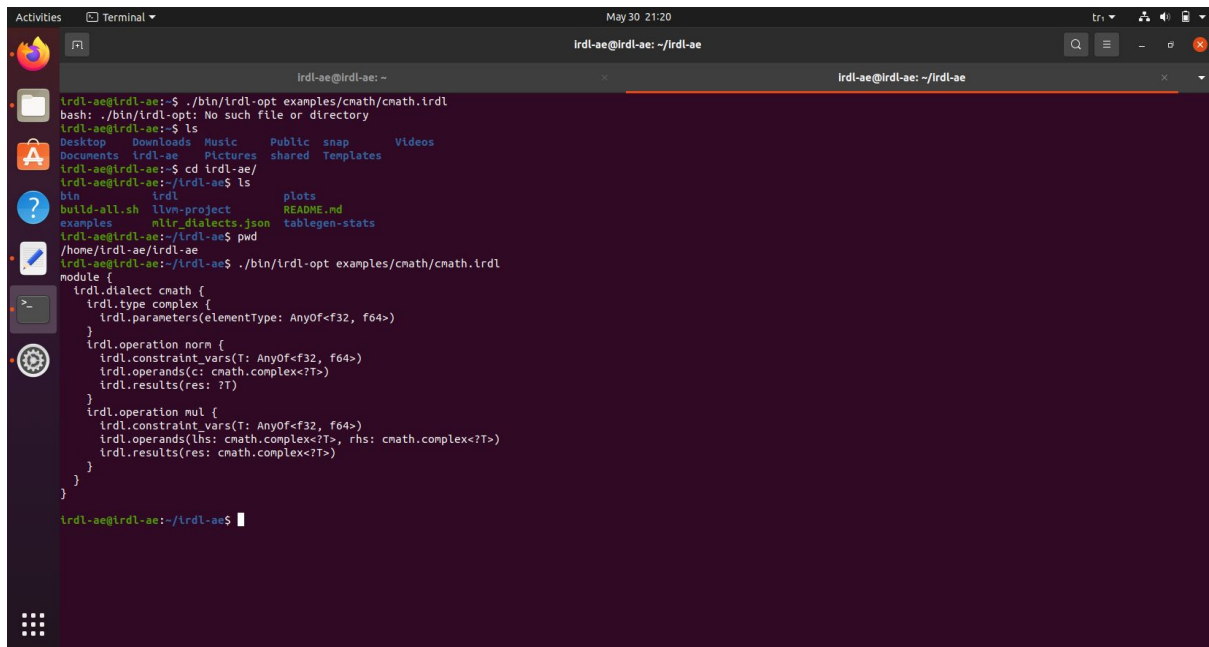
# Reproduction of Artifact

- General layout of the project



# Reproduction of Artifact

- Parsing cmath module with irdl-opt



A terminal window titled "Terminal" with a date and time of "May 30 21:20". The window shows the execution of several commands to reproduce an artifact. The user is in a directory named "irdl-ae" and runs the following commands:

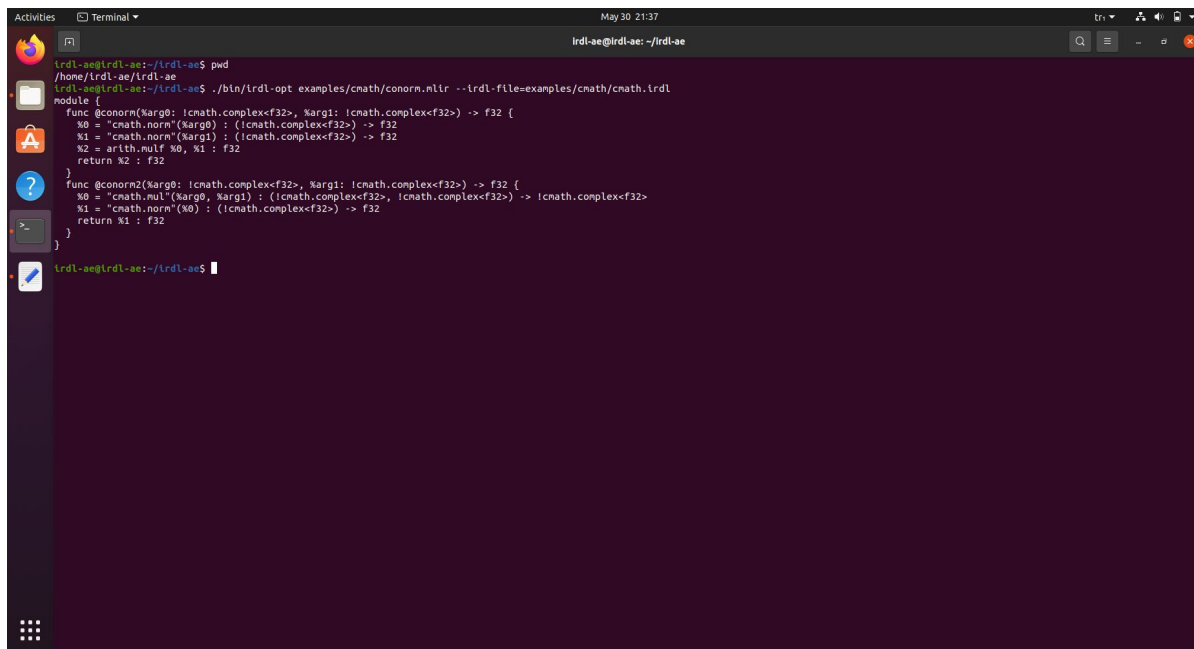
```
irdl-ae@irdl-ae:~$ ./bin/irdl-opt examples/cmath/cmath.irdl
bash: ./bin/irdl-opt: No such file or directory
irdl-ae@irdl-ae:~$ ls
Desktop  Downloads  Music      Public    snap      Videos
Documents  irdl-ae  Pictures  shared   Templates
irdl-ae@irdl-ae:~$ cd irdl-ae/
irdl-ae@irdl-ae:~/irdl-ae$ ls
bin      irdl      plots
build-all.sh  llvm-project  README.md
examples      nlr_dialects.json  tablegen-stats
irdl-ae@irdl-ae:~/irdl-ae$ pwd
/home/irdl-ae/irdl-ae
irdl-ae@irdl-ae:~/irdl-ae$ ./bin/irdl-opt examples/cmath/cmath.irdl
module {
  irdl.dialect cmath {
    irdl.type complex {
      irdl.parameters(elementType: AnyOf<f32, f64>)
    }
    irdl.operation norm {
      irdl.constraint_vars(T: AnyOf<f32, f64>)
      irdl.operands(c: cmath.complex<T>)
      irdl.results(res: ?T)
    }
    irdl.operation mul {
      irdl.constraint_vars(T: AnyOf<f32, f64>)
      irdl.operands(lhs: cmath.complex<?T>, rhs: cmath.complex<?T>)
      irdl.results(res: cmath.complex<?T>)
    }
  }
}
```

The terminal output shows the parsing of the cmath module, resulting in a module definition for the cmath dialect. The module contains two operations: norm and mul. The norm operation takes a complex number as input and returns a scalar of the same type. The mul operation takes two complex numbers as input and returns a complex number of the same type.



# Reproduction of Artifact

- Registering cmath dialect and parsing the conorm.mlir example



```
lrldl-ae@lrldl-ae:~/lrldl-ae$ pwd
/home/lrldl-ae/lrldl-ae
lrldl-ae@lrldl-ae:~/lrldl-ae$ ./bin/lrldl-opt examples/cmath/conorm.nllr --lrldl-file=examples/cmath/cmath.lrdl
module {
  func @conorm(%arg0: !cmath.complex<f32>, %arg1: !cmath.complex<f32>) -> f32 {
    %0 = "cnath.norm"(%arg0) : (!cmath.complex<f32>) -> f32
    %1 = "cnath.norm"(%arg1) : (!cmath.complex<f32>) -> f32
    %2 = arith.mulf %0, %1 : f32
    return %2 : f32
  }
  func @conorm2(%arg0: !cmath.complex<f32>, %arg1: !cmath.complex<f32>) -> f32 {
    %0 = "cnath.mul"(%arg0, %arg1) : (!cmath.complex<f32>, !cmath.complex<f32>) -> !cmath.complex<f32>
    %1 = "cnath.norm"(%0) : (!cmath.complex<f32>) -> f32
    return %1 : f32
  }
}
```

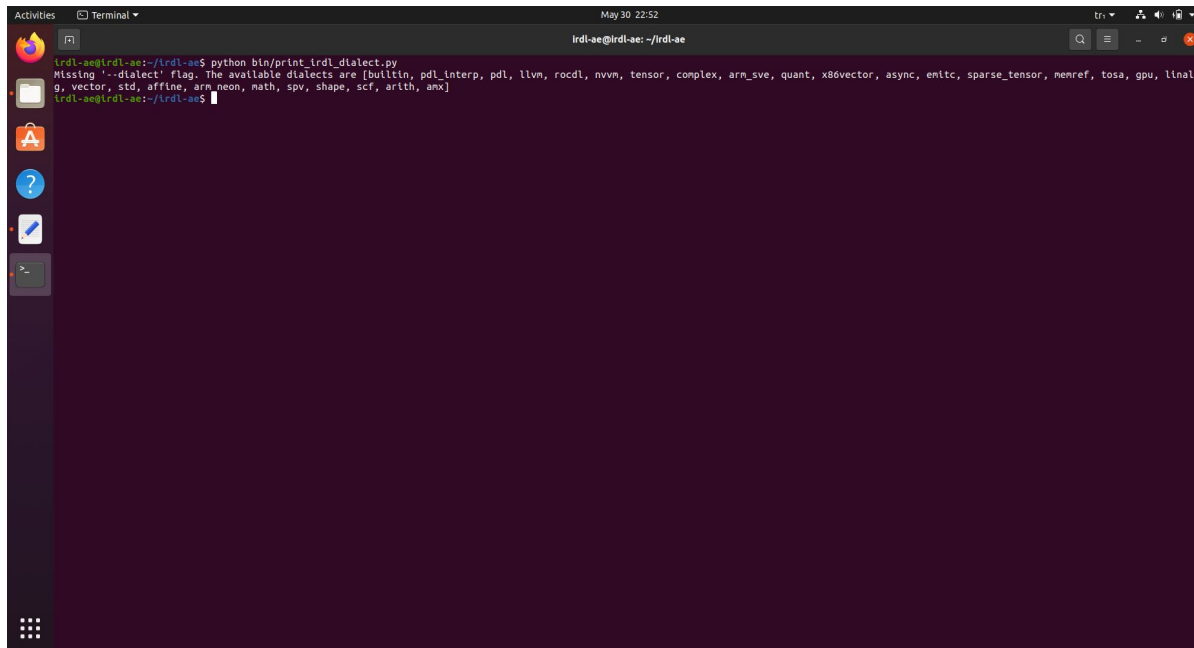
## Reproduction of Artifact

- Extracting MLIR dialects in order to convert them into IRDL format

[illegible]

# Reproduction of Artifact

- Querying the list of available dialects

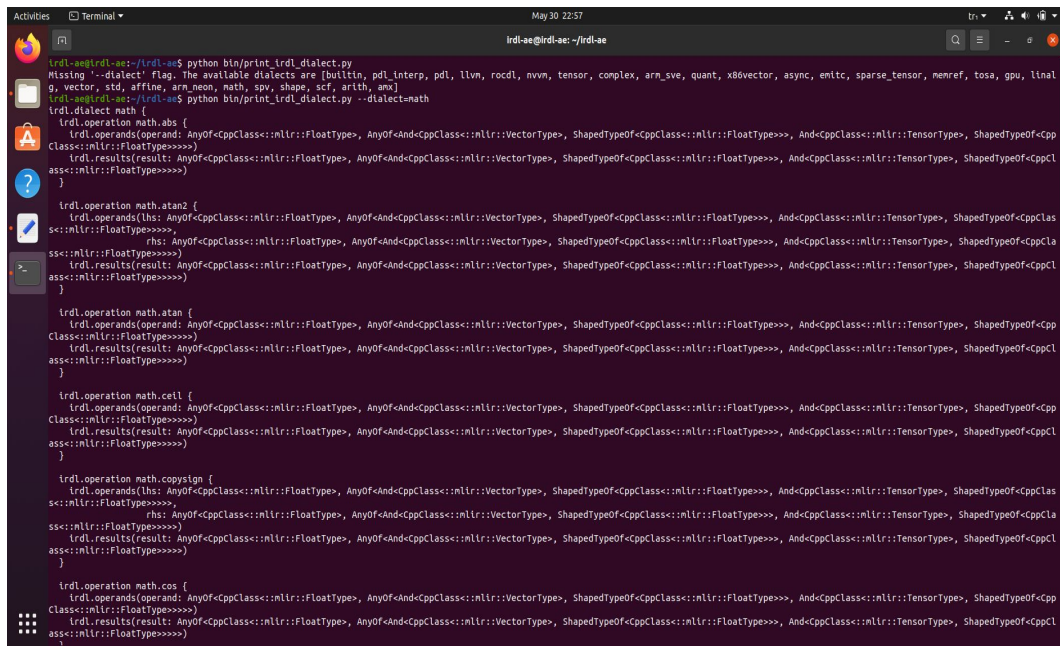


```
irdl-ae@irdl-ae:~/irdl-ae$ python bin/print_irdl_dialect.py
Missing '--dialect' flag. The available dialects are [builtin, pdl_interp, pdl, llvm, rocdl, nvvm, tensor, complex, arm_sve, quant, x86vector, async, emltc, sparse_tensor, memref, tosa, gpu, linalg, vector, std, affine, arm_neon, math, spv, shape, scf, arith, anx]
irdl-ae@irdl-ae:~/irdl-ae$
```

The image shows a terminal window with a dark background. The prompt is `irdl-ae@irdl-ae:~/irdl-ae$`. The command `python bin/print_irdl_dialect.py` has been executed. The output is a single line of text: `Missing '--dialect' flag. The available dialects are [builtin, pdl_interp, pdl, llvm, rocdl, nvvm, tensor, complex, arm_sve, quant, x86vector, async, emltc, sparse_tensor, memref, tosa, gpu, linalg, vector, std, affine, arm_neon, math, spv, shape, scf, arith, anx]`. The prompt is now `irdl-ae@irdl-ae:~/irdl-ae$`.

# Reproduction of Artifact

- Converting math dialect to IRDL format



A terminal window titled 'May 30 22:57' and 'irdl-ae@irdl-ae: ~/irdl-ae' displays the execution of a script to print the IRDL dialect. The script is run as `python bin/print_irdl_dialect.py`. The output shows the IRDL dialect definition, including the `math` dialect and several operations: `abs`, `atan2`, `atan`, `ceil`, `copysign`, and `cos`. Each operation is defined with its operands, results, and the types of its operands and results. The types are defined using the `AnyOf` and `And` classes, and the `ShapedTypeOf` class. The output is as follows:

```
irdl-ae@irdl-ae:~/irdl-ae$ python bin/print_irdl_dialect.py
Missing '--dialect' flag. The available dialects are [builtin, pdl, llvm, rocdl, nvvm, tensor, complex, arm_sve, quant, x86vector, async, emitc, sparse_tensor, nemref, tosa, gpu, linalg, vector, std, affine, arm_neon, math, spv, shape, scr, arith, amx]
irdl-ae@irdl-ae:~/irdl-ae$ python bin/print_irdl_dialect.py --dialect=math
irdl.dialect math {
  irdl.operation math.abs {
    irdl.operands(operand: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }

  irdl.operation math.atan2 {
    irdl.operands(lhs: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>,
    rhs: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }

  irdl.operation math.atan {
    irdl.operands(operand: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }

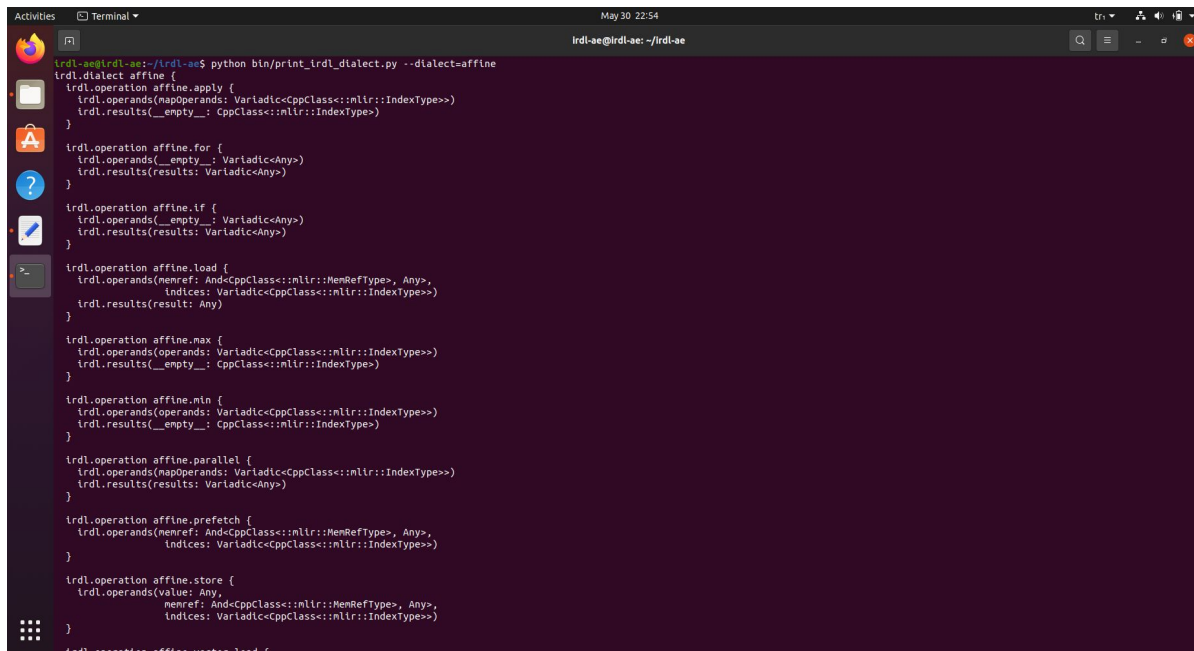
  irdl.operation math.ceil {
    irdl.operands(operand: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }

  irdl.operation math.copysign {
    irdl.operands(lhs: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>,
    rhs: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }

  irdl.operation math.cos {
    irdl.operands(operand: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
    irdl.results(result: AnyOf<CppClass::mlir::FloatType>, AnyOf<And<CppClass::mlir::VectorType>, ShapedTypeOf<CppClass::mlir::FloatType>,>, And<CppClass::mlir::TensorType>, ShapedTypeOf<CppClass::mlir::FloatType>)>
  }
}
```

# Reproduction of Artifact

- Converting math affine dialect to IRDL format



The screenshot shows a terminal window with the following content:

```
Activities Terminal May 30 22:54 trn  
irdl-ae@irdl-ae:~/irdl-ae$ python bin/print_irdl_dialect.py --dialect=affine  
irdl.dialect affine {  
  irdl.operation affine.apply {  
    irdl.operands(mapOperands: Variadic<CppClass::mlir::IndexType>)  
    irdl.results(_empty__: CppClass::mlir::IndexType)  
  }  
  
  irdl.operation affine.for {  
    irdl.operands(_empty__: Variadic<Any>)  
    irdl.results(results: Variadic<Any>)  
  }  
  
  irdl.operation affine.if {  
    irdl.operands(_empty__: Variadic<Any>)  
    irdl.results(results: Variadic<Any>)  
  }  
  
  irdl.operation affine.load {  
    irdl.operands(memref: And<CppClass::mlir::MemRefType, Any>,  
      indices: Variadic<CppClass::mlir::IndexType>)  
    irdl.results(result: Any)  
  }  
  
  irdl.operation affine.max {  
    irdl.operands(operands: Variadic<CppClass::mlir::IndexType>)  
    irdl.results(_empty__: CppClass::mlir::IndexType)  
  }  
  
  irdl.operation affine.min {  
    irdl.operands(operands: Variadic<CppClass::mlir::IndexType>)  
    irdl.results(_empty__: CppClass::mlir::IndexType)  
  }  
  
  irdl.operation affine.parallel {  
    irdl.operands(mapOperands: Variadic<CppClass::mlir::IndexType>)  
    irdl.results(results: Variadic<Any>)  
  }  
  
  irdl.operation affine.prefetch {  
    irdl.operands(memref: And<CppClass::mlir::MemRefType, Any>,  
      indices: Variadic<CppClass::mlir::IndexType>)  
  }  
  
  irdl.operation affine.store {  
    irdl.operands(value: Any,  
      memref: And<CppClass::mlir::MemRefType, Any>,  
      indices: Variadic<CppClass::mlir::IndexType>)  
  }  
  
  irdl.operation affine.vector.load {
```

# Reproduction of Artifact

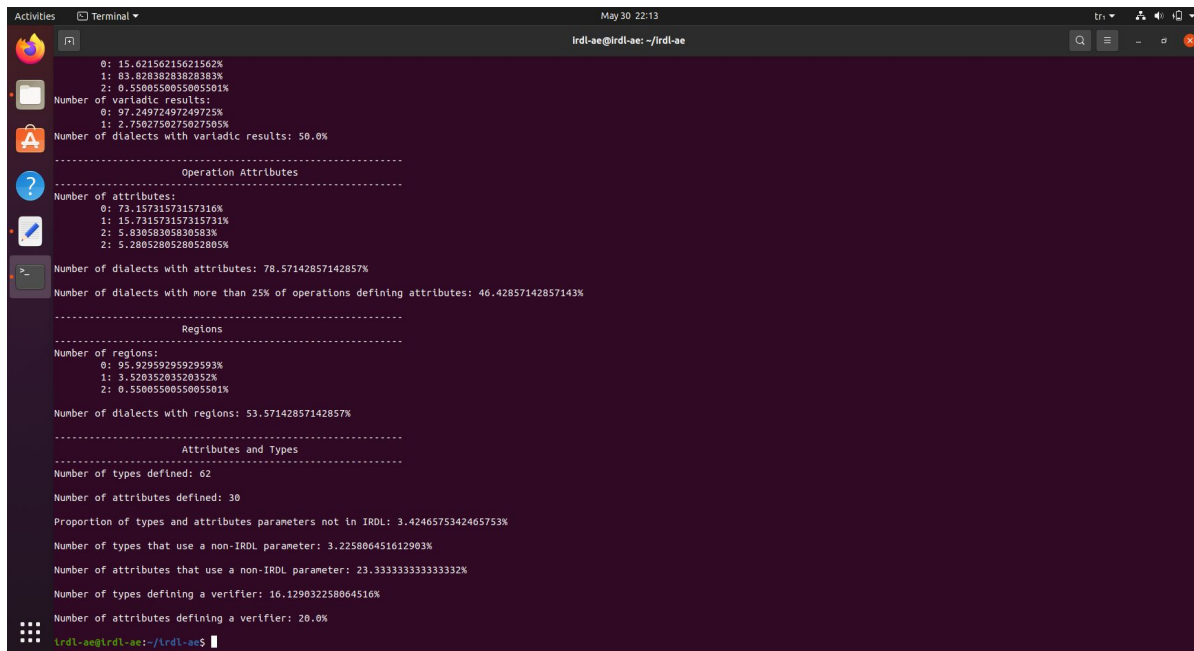
- Reproduced table statistics for Operations, Operands, Results, Operation Attributes

```
Activities Terminal May 30 22:12
lrld-ae@lrld-ae:~/lrld-ae$ mkdir -p plots && python bin/generate_paper_plots.py

-----
Operations
-----
Number of operations: 909
Number of operations requiring a verifier: 69.85698569856986%
Number of operations requiring IRDL-C++ for args: 2.8602860286028604%
-----
Operands
-----
Number of operands:
0: 11.66116611661166%
1: 40.5940594059406%
2: 32.01320132013201%
3+: 15.731573157315731%
Number of variadic operands:
0: 82.83028302830284%
1+: 17.16171617161716%
Number of dialects with variadic operands: 78.57142857142857%
Number of dialects with more than 25% of variadic operands: 46.42857142857143%
-----
Results
-----
Number of results:
0: 15.62156215621562%
1: 89.62028302830283%
2: 0.5500550055005501%
Number of variadic results:
0: 97.24972497249725%
1: 2.7502750275027504%
Number of dialects with variadic results: 50.0%
-----
Operation Attributes
-----
Number of attributes:
0: 73.15731573157316%
1: 15.731573157315731%
2: 5.83058305830583%
2: 5.2805280528052805%
Number of dialects with attributes: 78.57142857142857%
Number of dialects with more than 25% of operations defining attributes: 46.42857142857143%
```

# Reproduction of Artifact

- Reproduced table statistics for Regions and Number info for Types and Attributes



```
Activities Terminal May 30 22:13 trn irdl-ae@irdl-ae: ~/irdl-ae

0: 15.62156215621562%
1: 83.82838283828383%
2: 0.5500550055005501%
Number of variadic results:
0: 97.24972497249725%
1: 2.7502750275027505%
Number of dialects with variadic results: 50.0%

-----
Operation Attributes
-----
Number of attributes:
0: 73.15731573157316%
1: 15.731573157315731%
2: 5.83058305830583%
2: 5.2805280528052805%
Number of dialects with attributes: 78.57142857142857%
Number of dialects with more than 25% of operations defining attributes: 46.42857142857143%

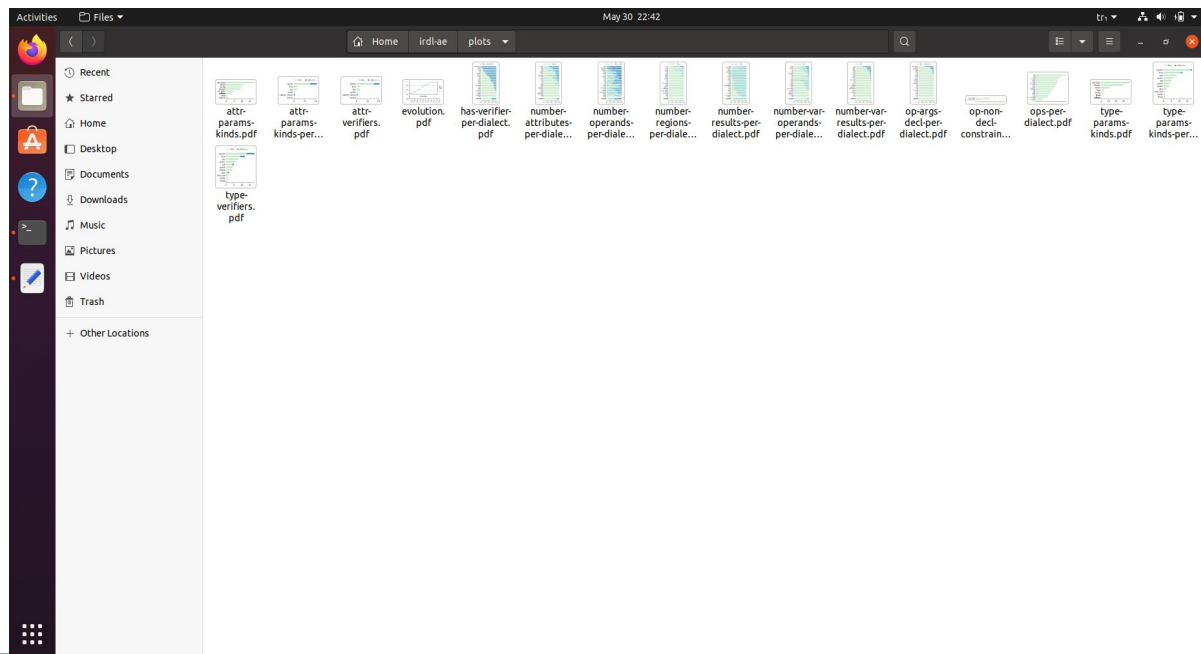
-----
Regions
-----
Number of regions:
0: 95.92959295929593%
1: 3.52035203520352%
2: 0.5500550055005501%
Number of dialects with regions: 53.57142857142857%

-----
Attributes and Types
-----
Number of types defined: 62
Number of attributes defined: 30
Proportion of types and attributes parameters not in IRDL: 3.4246575342465753%
Number of types that use a non-IRDL parameter: 3.225806451612903%
Number of attributes that use a non-IRDL parameter: 23.333333333333332%
Number of types defining a verifier: 16.129032258064516%
Number of attributes defining a verifier: 20.0%

irdl-ae@irdl-ae:~/irdl-ae$
```

# Reproduction of Artifact

- Plots are reproduced with command: `mkdir -p plots && python bin/generate_paper_plots.py`





# References

- <https://www.researchgate.net/profile/Peter-Fritzson/publication/228792639/figure/fig1/AS:393782852898820@1470896556105/Abstract-syntax-tree-of-the-while-loop.png>
- [https://raw.githubusercontent.com/Naios/notepad\\_llvm/master/preview.png](https://raw.githubusercontent.com/Naios/notepad_llvm/master/preview.png)
- <https://dl.acm.org/doi/pdf/10.1145/3519939.3523700>