Non-attribute property improvements in MLIR

[name removed]

July 15, 2025

Abstract

Over the past year, many improvements have been made to the infrastructure for using non-attribute properties in MLIR. Such properties allow constants and other data to be stored inline in the operation struct without potentially-expensive attribute uniquing.

This presentation will showcase these improvements, such as new support for such properties in TableGen-based builder/parser/printer generation, operation verification, improved generic builders, and declarative rewrite rules. It will also present future directions for non-attribute properties.

Talk length 5-10 minutes (lightning or quick)

1 Extended abstract

I intend to present the work I have done and will do to enable non-attribute property support in MLIR and MLIR's TableGen infrastructure. This work has been ongoing for the past year, but may have gone unnoticed within the wider community.

For example, my PR #94732 added support for using non-attribute properties in TableGen operation definitions, including parser and printer generation. With that PR, you could use non-attribute properties much like you would use an attribute, as in:

```
def ExampleOp : MyDialectOp<"example">,
  Arguments<(I32Prop:$x,
    OptionalProp<I64ArrayProp>:$y,
    UnitProp:$flag> {
  let assemblyFormat = "$x ($y^)? $(`flag` $flag^)?";
}
```

which allows for syntax like

```
my_dialect.example 4
my_dialect.example 2 [-1, -2] flag
```

and generates builders such as

```
build(OperationState, Location, int32_t x,
  std::optional<ArrayRef<int64_t>> y = std::nullopt,
  bool flag = false);
```

, which added much-needed ergonomic improvements to non-attribute properties, which were needed to alleviate the need to use a custom directive when defining syntax for them. This code was also used to improve the usage of such properties in the LLVM dialect (mainly around overflow flags).

This sequence of ergonomic improvements continued with my PR #120176 and its followups like PR #140848 and PR #140849. These PRs added a predicate field to properties, allowing for autogenerating verification constraints for such properties. For example, one can now have an `NonNegativeI64Prop:$x` or even an `ArrayProp<NonNegativeI64Prop>:$a` and have the constraint for the property (or each element of it) appear in the verifier.

PRs #132148 and #132149 improved enum generation. Now, the definition of an enum is decoupled from defining an attribute that holds it, and these C++ enums can easily be used in the `EnumProp` class, allowing easy autogenerated syntax like `overflow<nsw, nuw>` (currently used in the LLVM dialect).

Other parts of the infrastructure have also been improved, such as with my PR#124113, which adds forms of the generic builder (which classically took the result types, operands, and attribute dictionary) that separate the inherent attributes and properties (in an opaque properties struct) from discardable attributes. Similarly, my PR #143071 adds support for non-attribute properties in the declarative rewrite rule infrastructure, where trying to define rewrite rules on such properties would previously crash. That is, one can now have rewrites such as this one (taken from MLIR's tests)

```
def : Pat<
  (TestPropPatternOp1
    $tag,
    NonNegativeI64Prop:$val,
    ConstantProp<BoolProp, "false">),
  (TestPropPatternOp1
    $tag,
    (NativeCodeCall<"$0 + 1"> $val),
    ConstantProp<BoolProp, "true">)>;
```

This work to make non-attribute properties ergonomic and useful is ongoing. I will, if time permits, propose future work in this direction, such as

- Support for strict inherent/discardable attribute separation in operation creation
- Improvements to property parsing and `prop-dict`
- The merits and risks of switching applicable inherent attributes to properties throughout upstream MLIR (the prop-ocalypse)

I expect that improvements to the usability of such properties and their increased adoption will also improve the performance of MLIR-based compilers through removing unneeded attribute-uniquing steps, though I do not yet have benchmarks for this yet.

The presentation will necessarily be a short overview of the topic, especially if it's accepted as a lightning talk, but will serve as an introduction to work I've done in the area.