Notes on non-attribute properties

Krzysztof Drewniak[1]

Advanced Micro Devices

Aug 14, 2025

[1]Krzysztof.Drewniak@amd.com

What are non-attribute properties

- Ability to add non-attribute values directly to operations
- Not uniqued
- Mutable
- Still an evolving piece of MLIR infrastructure
- Defineable in tablegen
- Have an interface type (passed to builder etc.) and storage type (stored in properties struct)
- Convertable to/from attributes

Example op

```
def MyOp : MyDialect<"my_op"> {
  let arguments = (ins
    I64Attr:$attr,
    I64Prop:$a,
    StringProp:$b,
    DefaultValuedProp<I64Prop, "0">:$c
  );
}
```

Properties struct

```
class MyOpGenericAdaptorBase {
struct Properties {
  IntegerAttr attr;
  int64_t a;
  std::string b;
  int64_t c = 0;

  IntegerAttr getAttr() const;
  void setAttr(IntegerAttr value);
  int64_t getA() const;
  ...
  StringRef getB() const;
  void setB(StringRef value);
}; ...
```

All ops generated upstream have a `Properties` struct today!
(We retain the option to not use one for inherent attributes when TableGen'ing and should deprecate that.)

Builders

```
MyOp::create(OpBuilder&, Location,
  IntegerAttr attr, int64_t a, StringRef b,
  int64_t c = 0);
...
MyOp::create(OpBuilder&, Location,
  TypeRange resultTypes, ValueRange operands,
  const Properties& properties,
  ArrayRef<NamedAttribute> discardableAttrs);
```

Second form is new, not fully efficient yet.

Example syntaxes

```
"my_dialect.my_op"() <{attr = 1 : i64,
  a = 2, b = "x", c = 0 : i64} {} : () -> ()

asmFormat = "attr-dict $attr $a $b $c";
my_dialect.my_op 1 2 "x" 0

asmFormat = "attr-dict $attr $a $b ($c^)";
my_dialect.my_op 1 2 "x"

asmFormat = "prop-dict attr-dict";
my_dialct.my_op <{attr = 1 : i64, ... c = 0 : i64}>

# But I think it should be
my_dialect.my_op <attr = 1, a = 2, b = "x">
```

Printing and parsing

- Non-attribute properties work in assembly formats.
- Optional/default parsing supported `$c` vs `($c^)?`.
- Not listing a non-attribute property means needing a `prop-dict` directive.
- `prop-dict` prints all non-elided **inherent** attributes and non-attributes properties as an attribute — like in generic form
- Using prop-dict means attr-dict is discardable attributes only (not currently taken advantage of)

Missing infrastructure — op creation

- Operation::create and OperationState flow that separates properties and discardable attributes.
- Currently, we always scan the attribute list to pull out inherent attributes
- Need some opt-in way to not do that
- Details being hashed out on forum

Missing infrastructure — op parsing/printing

- Proposed change to `prop-dict` in assembly formats.
- Would make prop-dict like a `struct` of all inherent properties not mentioned elsewhere
- Could be changed backwards-compatibly for most cases – just check for a `DictionaryAttr` after the `<`.

Missing? — getPropertyAsAttr

- Have `getPropertiesAsAttr` and `setPropertiesFromAttr` for generic parsing and current `prop-dict`
- Should we add `getPropertyAsAttr` and `setPropertyFromAttr`?
- Take existing `get/setInherentAttr()` and add code for non-attribute properties
- Pros: Allows reasonably generic code, introspection, may simplify bindings
- Cons: Might lead to people using it when they shouldn't

FFI bindings

- Currently — no way to get/set non-attribute properties from C, Python, etc.
- Let's discuss answers
- IMO, the C++ types normally used for non-attribute properties can easily become C types
- So - tablegen C versions of Properties struct
- But autogen'd C headers have been objected to
- Python ... not my wheelhouse, apparently can wrap C++ directly?

Summary

- Non-attribute property ifrastructure has evolved, is continuing
- Still have infrastructure questions
- But also ... bigger questions about what a non-attribute property *should be*.
- And so I'll pass it to Fabian