MLIR properties, design discussion and next steps

Open Design Meeting

Hosted: Fabian Mora Cordero
Participants: Mehdi Amini, Krzysztof Drewniak

Background

Attributes

“Attributes are the mechanism for specifying constant data on operations in places where a variable is never allowed - e.g. the comparison predicate of a cmpi operation. Each operation has an attribute dictionary, which associates a set of attribute names to attribute values…”

MLIR LangRef

Characteristics:

- Lifetime bound and owned by the MLIR context
- Unique wrt the context
- Definition owned by dialects
- Can be parsed and printed generically
- Can have interfaces

MLIR properties

[RFC] Introducing MLIR Operation Properties

Mehdi Amini

“Properties are extra data members stored directly on an Operation class. They provide a way to store inherent attributes and other arbitrary data. The semantics of the data is specific to a given operation, and may be exposed through Interfaces accessors and other methods. Properties can always be serialized to Attribute in order to be printed generically.”

MLIR LangRef

```
// Any kind of integer stored as properties.
class IntProp<string storageTypeParam, string desc =
""> : Property<storageTypeParam, desc> {
 let summary = “...”
 let optionalParser = [{...}];
 let printer = "...";
 let writeToMlirBytecode = [{...}];
 let readFromMlirBytecode = [{...}];
 code convertToAttribute = [{...}];
 code convertFromAttribute = [{...}];
}

def I32Prop : IntProp<"int32_t">;
```

Op example:

```
def MyOp : Dialect_Op<"my_op"> {
 let arguments = (ins UnitProp:$prop, UnitAttr:$attr);
 let assemblyFormat = [{ attr-dict }];
}
```

TableGen spec

```
struct Properties {
 using attrTy = ::mlir::UnitAttr;
 attrTy attr;
 auto getAttr() const {
   auto &propStorage = this->attr;
   return
::llvm::dyn_cast_or_null<::mlir::UnitAttr>(propStorage)
;
 }
 void setAttr(const ::mlir::UnitAttr &propValue) {
   this->attr = propValue;
 }
 using propTy = bool;
 propTy prop = false;
};
```

C++ declaration

```
::mlir::Attribute MyOp::getPropertiesAsAttr(::mlir::MLIRContext *ctx,
const Properties &prop) {
   ::mlir::SmallVector<::mlir::NamedAttribute> attrs;
   ::mlir::Builder odsBuilder{ctx};
     const auto &propStorage = prop.attr;
     if (propStorage)
       attrs.push_back(odsBuilder.getNamedAttr("attr",
                                               propStorage));
   {
     const auto &propStorage = prop.prop;
     auto attr = [&]() -> ::mlir::Attribute {
       if (propStorage)
     return ::mlir::UnitAttr::get(ctx);
   else
     return ::mlir::BoolAttr::get(ctx, false);
     }();
     attrs.push_back(odsBuilder.getNamedAttr("prop", attr));
   }
 if (!attrs.empty())
   return odsBuilder.getDictionaryAttr(attrs);
 return {};
}
```

C++ methods

Issues

- They broke C and python bindings: [mlir][python] Op properties are broken for python · Issue #150009 · llvm/llvm-project
- They are currently a C++ implementation detail of operations
  - No generic printing and parsing of props, as in the case of attrs or types
  - Their semantic meaning is subjugated to a C++ detail of ops, and not the IR
- They need attributes to interact generically with other components, eg. printing and parsing generic ops
  - This implementation is inefficient
- No interfaces, so not a full replacement for attributes in ops

Proposal

Amend their definition in the LangRef

"Properties are extra data members stored directly on an Operation class. They provide a way to store inherent attributes and other arbitrary data. The semantics of the data is specific to a given operation, and may be exposed through Interfaces accessors and other methods. Properties can always be serialized to Attribute in order to be printed generically."

MLIR LangRef

"Properties are a mechanism for specifying arbitrary mutable or immutable data on operations. The full semantics of the data are specific to a given property and operation. Each operation has a static property dictionary, associating names to properties."

MLIR LangRef

Long term implementation changes

- Remove round-tripping properties through attributes as a constraint
    - It’s wasteful and defeat the efficiency goal of props
    - It’s an excuse to avoid solving the underlying technical debt
- Add property verifiers
    - Needed for safe generic parsing and printing
    - These should be nop on full release mode, and there should be an option to disable them on runtime
- Add generic parsing and printing hooks
    - `&i32<0>`, `&gpu::binary<”...”>`, …
    - Removes the need to round-trip through attributes for generic parsing and printing

Long term implementation changes

- Add a `UniquePropStorage` to store opaque properties
  - Needed for parsing unknown generic properties
- This should serve as a stopgap measure for the C/Python bindings issue
- Add a `OpaquePropRef` class to hold references to opaque properties
  - Needed to have a `TypeID` safe way to interact with props in Ops
- Add a `PropRef` template class to hold concrete instances of prop refs
- Allow prop interfaces
- Make all attributes convertible to props, but not the other way

What about attrs? What’s the official guideline?

- Attributes should be used when:
    - Data rarely changes during the lifetime of the context
    - Fast-comparison is needed between data
    - The data should be persistent till the end of the context
    - Example: An attribute containing information for configuring an immutable pass pipeline
- Use properties in almost all cases except in those cases suggested by the attribute guidance

Optional changes

- Add a discardable prop dict and replace the discardable attr dict
  - This should be possible via a map and `UniquePropStorage`
- Remove unregistered ops
  - Seem like a relic of the past
  - Their interaction with props is limited