tensorflow / **tensorflow** Public

Code | Issues 2.1k | Pull requests 283 | Actions | Projects 1 | ...

a1320ec1ea

tensorflow / tensorflow / core / framework / **full_type_util.cc**

Dan Moldovan Define type information for mutex ops. ... ✗ History

1 contributor

132 lines (105 sloc) | 4.15 KB

```
/* Copyright 2020 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
==============================================================================*/

#include "tensorflow/core/framework/full_type_util.h"

#include "tensorflow/core/framework/attr_value.pb.h"
#include "tensorflow/core/framework/full_type.pb.h"
#include "tensorflow/core/framework/node_def.pb.h"
#include "tensorflow/core/framework/node_def_util.h"
#include "tensorflow/core/framework/op_def.pb.h"
#include "tensorflow/core/framework/types.h"
#include "tensorflow/core/platform/statusor.h"

namespace tensorflow {

namespace full_type {

```

```
OpTypeConstructor Nullary(FullTypeId t) {
  return [t](OpDef* op_def) {
    FullTypeDef* tdef =
        op_def->mutable_output_arg(0)->mutable_experimental_full_type();
    tdef->set_type_id(t);
    return Status::OK();
  };
}

OpTypeConstructor Unary(FullTypeId t, const string& var_name) {
  return [t, var_name](OpDef* op_def) {
    FullTypeDef* tdef =
        op_def->mutable_output_arg(0)->mutable_experimental_full_type();
    tdef->set_type_id(t);

    FullTypeDef* arg = tdef->add_args();
    arg->set_type_id(TFT_VAR);
    arg->set_s(var_name);

    return Status::OK();
  };
}

OpTypeConstructor UnaryGeneric(FullTypeId t) {
  return [t](OpDef* op_def) {
    FullTypeDef* tdef =
        op_def->mutable_output_arg(0)->mutable_experimental_full_type();
    tdef->set_type_id(t);

    FullTypeDef* arg = tdef->add_args();
    arg->set_type_id(TFT_ANY);

    return Status::OK();
  };
}

OpTypeConstructor UnaryTensorContainer(FullTypeId t, FullTypeId dtype) {
  return [t, dtype](OpDef* op_def) {
    FullTypeDef* tdef =
        op_def->mutable_output_arg(0)->mutable_experimental_full_type();
    tdef->set_type_id(t);

    FullTypeDef* arg = tdef->add_args();
    arg->set_type_id(TFT_TENSOR);
    FullTypeDef* targ = arg->add_args();
    targ->set_type_id(dtype);

    return Status::OK();
  };
```

```
}

StatusOr<FullTypeDef> SpecializeType(const AttrSlice& attrs,
                                     const OpDef& op_def) {
  FullTypeDef ft;
  ft.set_type_id(TFT_PRODUCT);

  for (int i = 0; i < op_def.output_arg_size(); i++) {
    auto* t = ft.add_args();

    *t = op_def.output_arg(i).experimental_full_type();

    // Resolve dependent types. The convention for op registrations is to use
    // attributes as type variables.
    // See https://www.tensorflow.org/guide/create_op#type_polymorphism.
    // Once the op signature can be defined entirely in FullType, this
    // convention can be deprecated.
    //
    // Note: While this code performs some basic verifications, it generally
    // assumes consistent op defs and attributes. If more complete
    // verifications are needed, they should be done by separately, and in a
    // way that can be reused for type inference.
    for (int j = 0; j < t->args_size(); j++) {
      auto* arg = t->mutable_args(i);
      if (arg->type_id() == TFT_VAR) {
        const auto* attr = attrs.Find(arg->s());
        DCHECK(attr != nullptr);
        if (attr->value_case() == AttrValue::kList) {
          const auto& attr_list = attr->list();
          arg->set_type_id(TFT_PRODUCT);
          for (int i = 0; i < attr_list.type_size(); i++) {
            map_dtype_to_tensor(attr_list.type(i), arg->add_args());
          }

        } else if (attr->value_case() == AttrValue::kType) {
          map_dtype_to_tensor(attr->type(), arg);

        } else {
          return Status(error::UNIMPLEMENTED,
                        absl::StrCat("unknown attribute type",
                                     attrs.DebugString(), " key=", arg->s()));
        }

        arg->clear_s();
      }
    }
  }

  return ft;
```

```
}

}  // namespace full_type

}  // namespace tensorflow
```