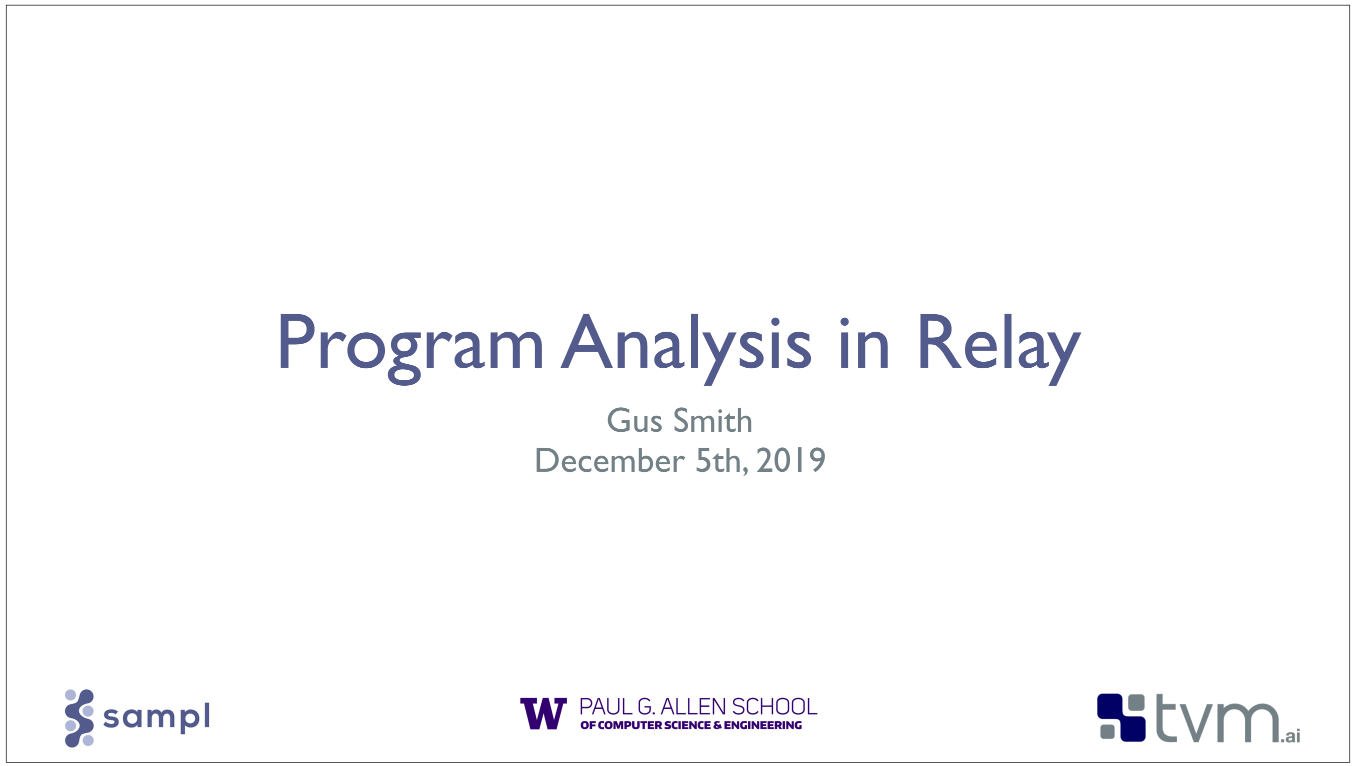
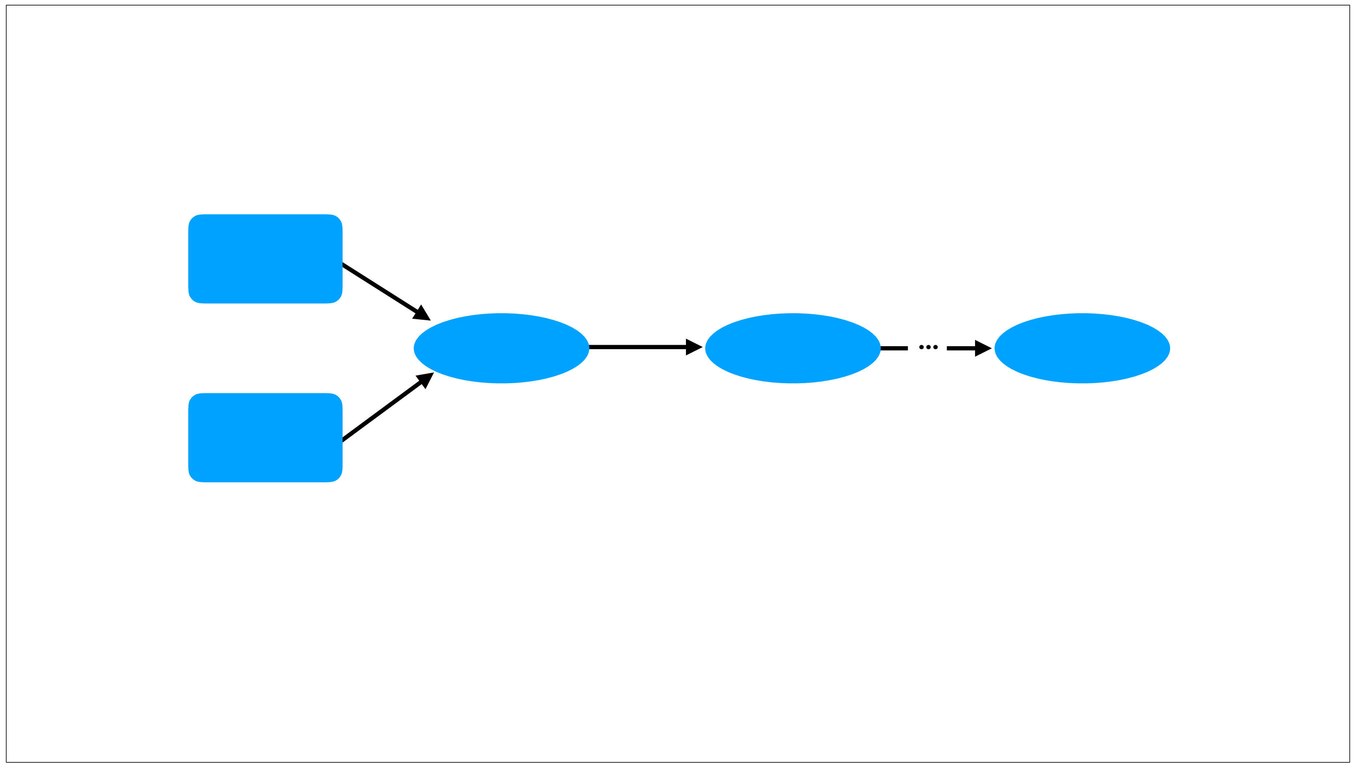


# Program Analysis in Relay

Gus Smith
December 5th, 2019

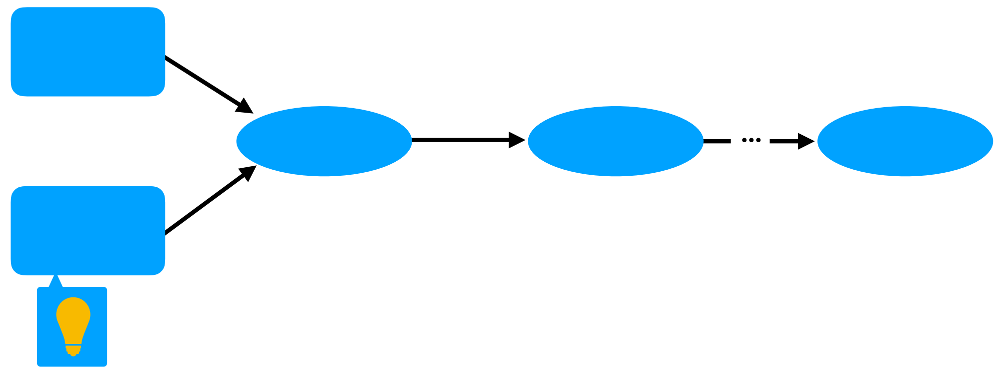
sampl
PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING
tvm.ai

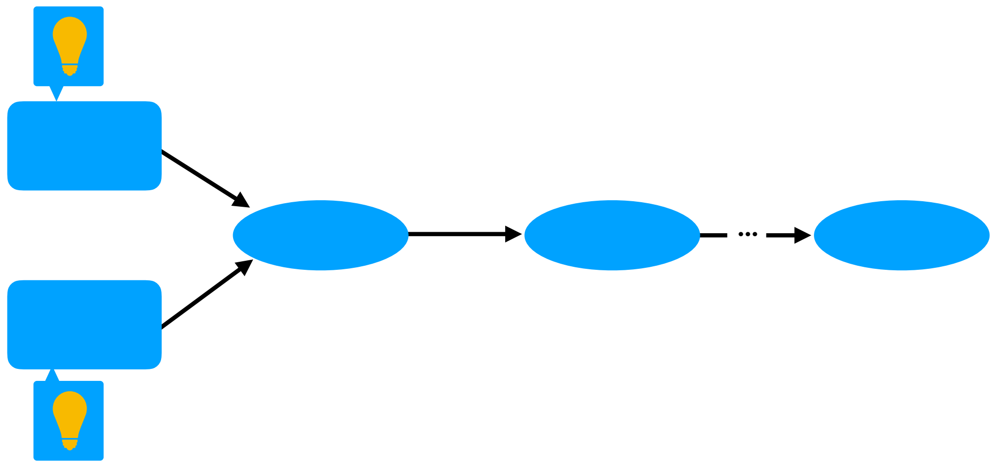
Hi everyone, my name's Gus Smith, from University of Washington's SAMPL lab.

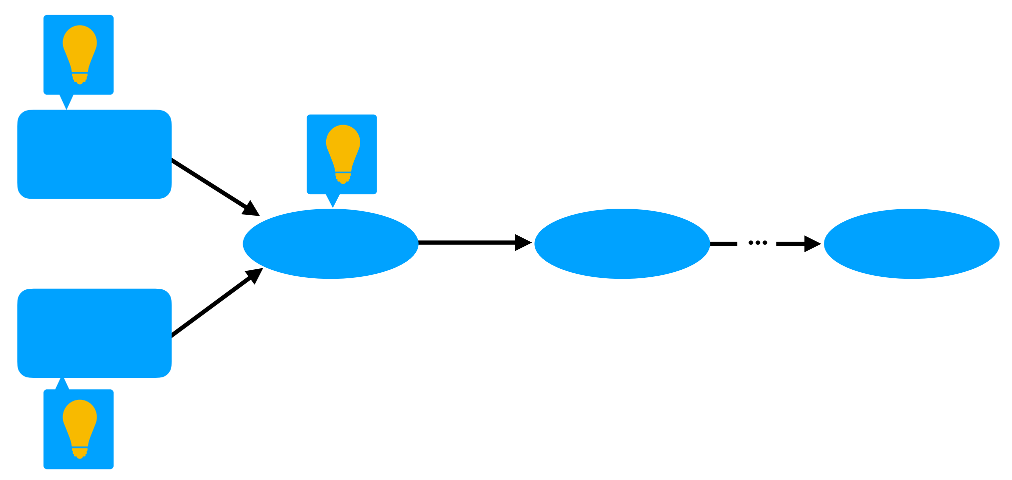


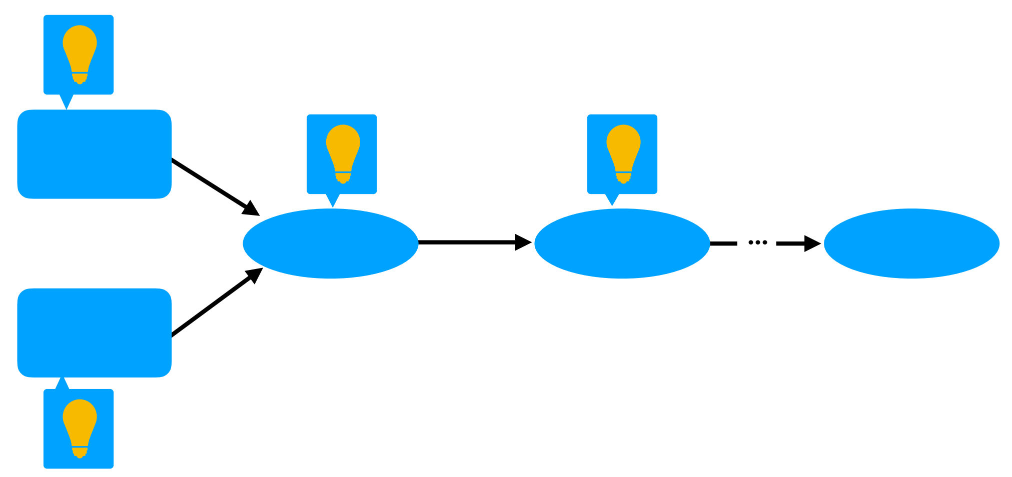
This past summer at Microsoft, I built a program analysis framework in Relay, useful for
[build lightbulbs] building simple static dataflow analyses and
[build summary square] putting their results into human-readable formats.

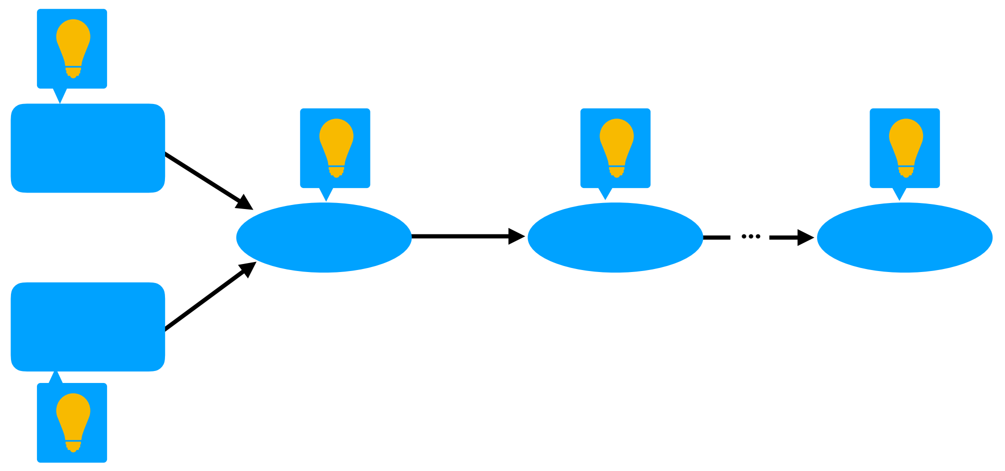
However, when I went to merge the framework back into Relay, I didn't find any appropriate place for it!

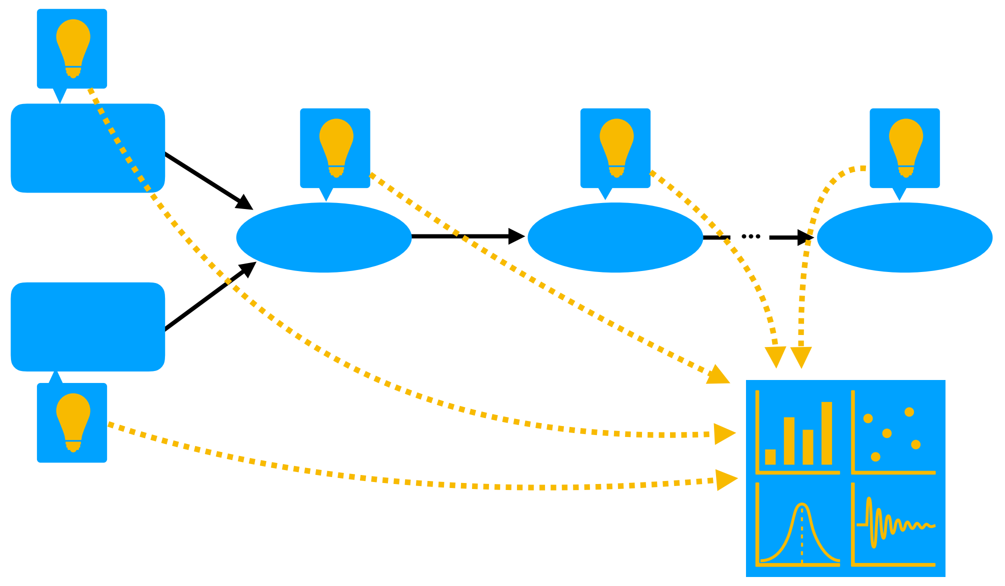












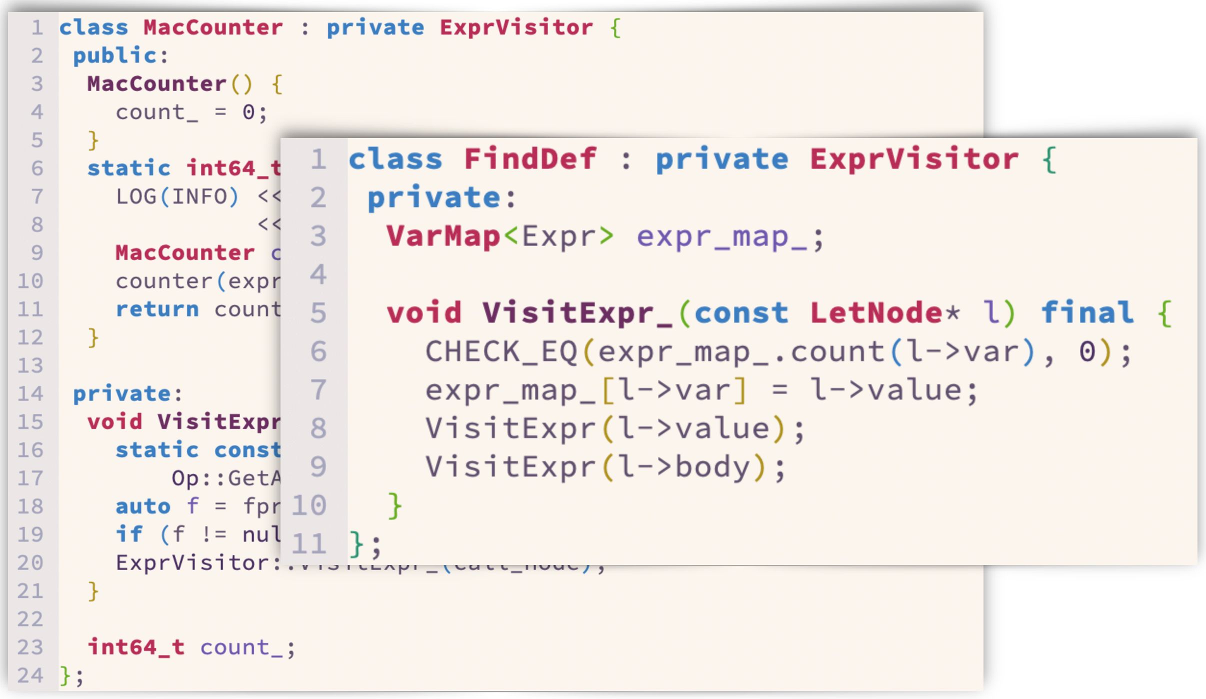
I was surprised by this—digging around in Relay, I saw program analyses of many different
[build] shapes and
[build] sizes, from simple static analyses like these, to complex dynamic analyses, such as those needed for quantization.

I discovered that there was no obvious place for my analysis framework because there is little to no analysis tooling in Relay at the moment.
These analyses are all built in an ad-hoc fashion, from the ground up—there's no underlying analysis framework which they rely on.

```cpp
class MacCounter : private ExprVisitor {
 public:
  MacCounter() {
    count_ = 0;
  }
  static int64_t GetTotalMacNumber(const Expr& expr) {
    LOG(INFO) << "This pass only counts MACs in direct CONV 2D, "
              << "CONV 2D Transpose and Dense ops";
    MacCounter counter;
    counter(expr);
    return counter.count_;
  }

 private:
  void VisitExpr_(const CallNode* call_node) final {
    static const auto& fprep =
        Op::GetAttr<FMacCount>("FMacCount");
    auto f = fprep.get(call_node->op, nullptr);
    if (f != nullptr) count_ += f(GetRef<Call>(call_node));
    ExprVisitor::VisitExpr_(call_node);
  }

  int64_t count_;
};
```



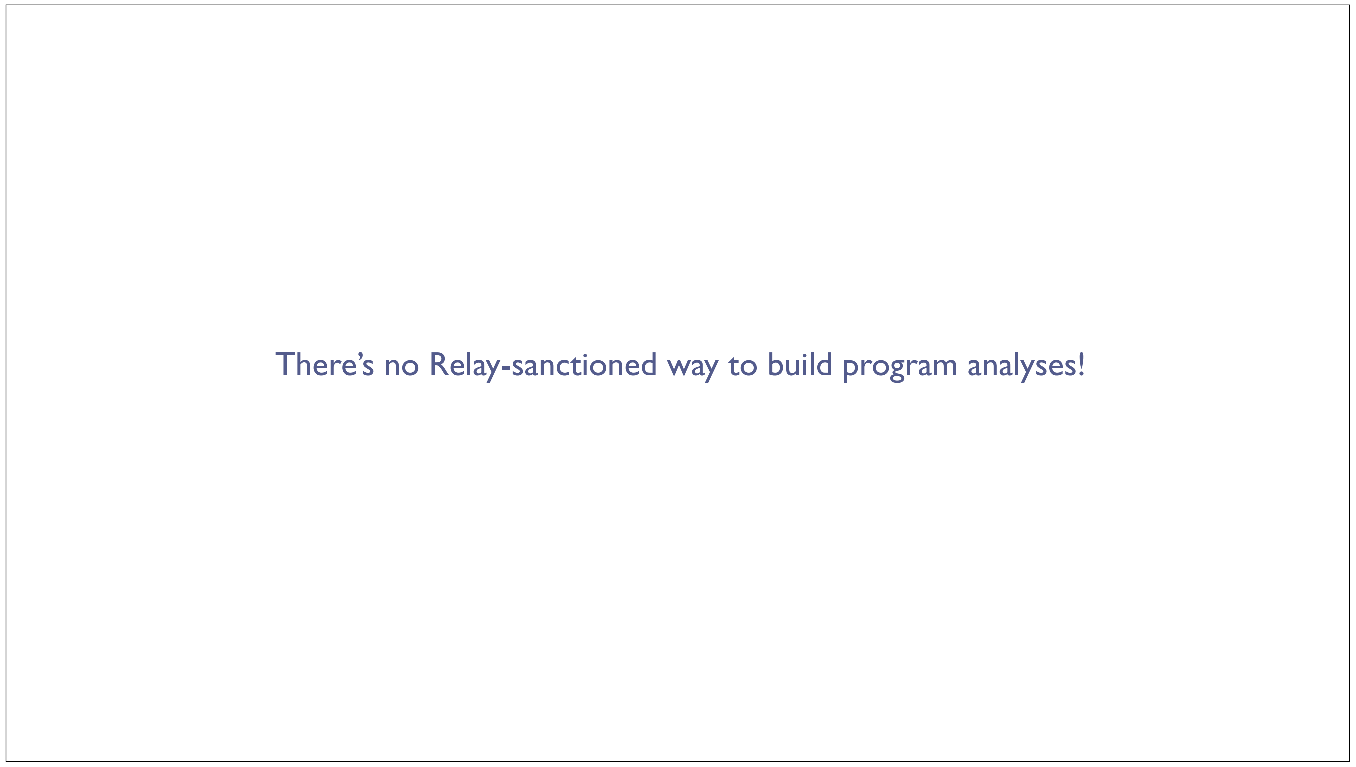
```
class MacCounter : private ExprVisitor {
 public:
  MacCounter() {
    count_ = 0;
  }
  static int64_t
    LOG(INFO) <<
              <<
    MacCounter c
    counter(expr
    return count
  }

 private:
  void VisitExpr
    static const
        Op::GetA
    auto f = fpr
    if (f != nul
    ExprVisitor::VisitExpr_(call_node);
  }

  int64_t count_;
};
```

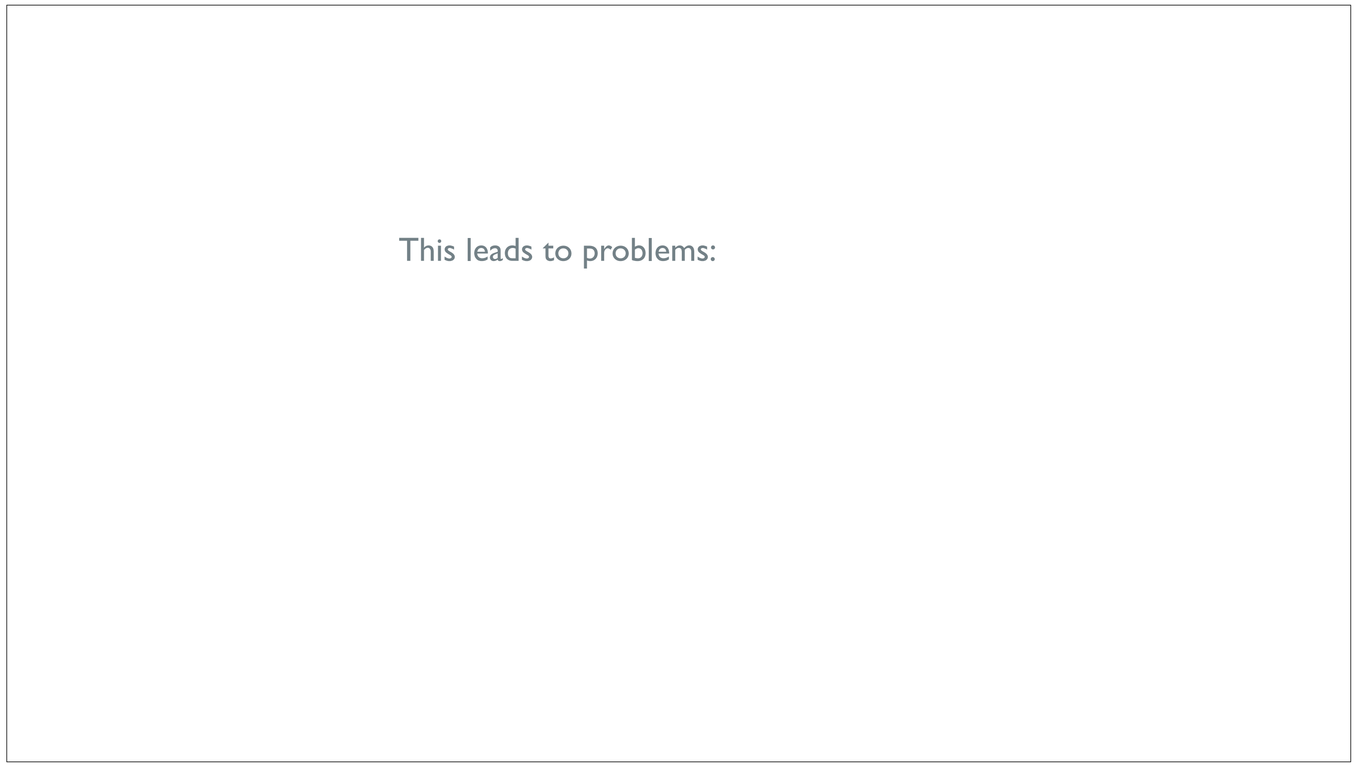
```
class FindDef : private ExprVisitor {
 private:
  VarMap<Expr> expr_map_;

  void VisitExpr_(const LetNode* l) final {
    CHECK_EQ(expr_map_.count(l->var), 0);
    expr_map_[l->var] = l->value;
    VisitExpr(l->value);
    VisitExpr(l->body);
  }
};
```


There's no Relay-sanctioned way to build program analyses!

This means that there is no Relay-sanctioned way to build program analyses.

Relay has all of the necessary tools to make program analyses of various flavors, but the tools are fairly basic, meaning analyses must be built up from basics each time.



This leads to problems:

This lack of an idiomatic way to build analyses can lead to problems.
[build] There can be duplication of effort, if basic analysis tools must be rewritten for each analysis.
[build] There's a high barrier to entry for new developers, who are expected to build their analyses using low-level APIs.
[build] And lastly, complex analyses, which may be written by just one or two people, are much less readable and maintainable by the broader community if they're built entirely from scratch.

This leads to problems:

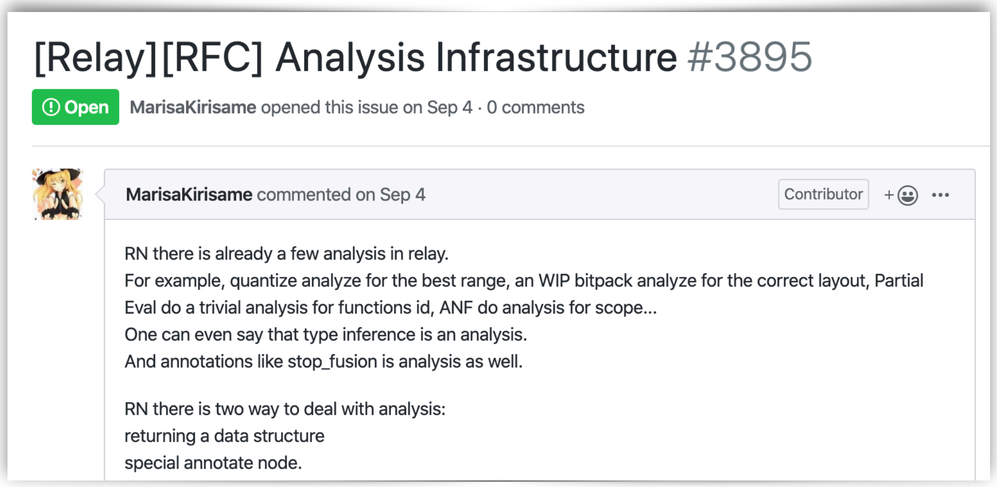
- Duplication of effort

This leads to problems:

- Duplication of effort
- High barrier to entry for new developers

This leads to problems:

- Duplication of effort
- High barrier to entry for new developers
- Less readability and maintainability


[Relay][RFC] Analysis Infrastructure #3895

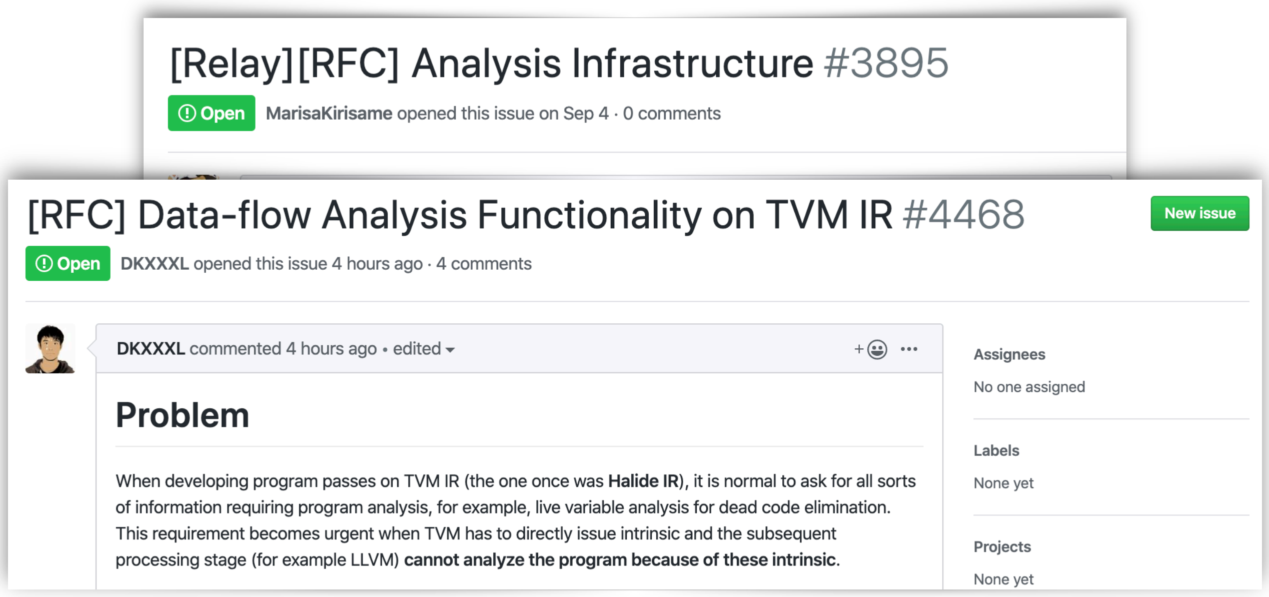
Open MarisaKirisame opened this issue on Sep 4 · 0 comments

MarisaKirisame commented on Sep 4 Contributor

RN there is already a few analysis in relay.
For example, quantize analyze for the best range, an WIP bitpack analyze for the correct layout, Partial Eval do a trivial analysis for functions id, ANF do analysis for scope...
One can even say that type inference is an analysis.
And annotations like stop_fusion is analysis as well.

RN there is two way to deal with analysis:
returning a data structure
special annotate node.

In addition to there being a *need* for better analysis tooling, there's also a community desire for better analysis tooling.
Recently, Marisa opened an RFC regarding analysis infrastructure in Relay.
[build] In addition, just yesterday, there was an RFC posted about dataflow analyses over TVM IR.
So clearly there's a desire to have one (or a few) "approved ways" to do analysis.


[Relay][RFC] Analysis Infrastructure #3895

Open MarisaKirisame opened this issue on Sep 4 · 0 comments

[RFC] Data-flow Analysis Functionality on TVM IR #4468

New issue

Open DKXXXL opened this issue 4 hours ago · 4 comments

DKXXXL commented 4 hours ago • edited

## Problem

When developing program passes on TVM IR (the one once was **Halide IR**), it is normal to ask for all sorts of information requiring program analysis, for example, live variable analysis for dead code elimination. This requirement becomes urgent when TVM has to directly issue intrinsic and the subsequent processing stage (for example LLVM) **cannot analyze the program because of these intrinsic**.

Assignees

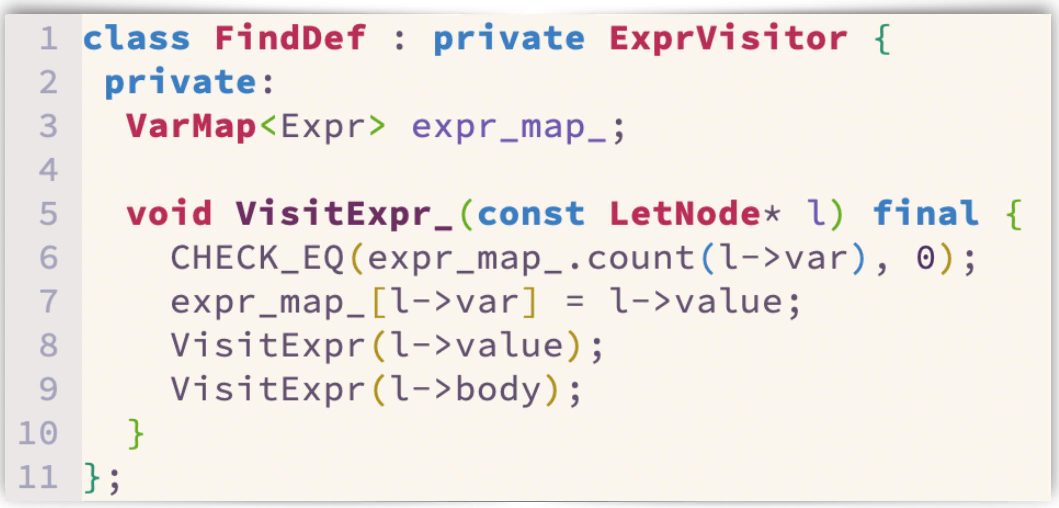
No one assigned

Labels

None yet

Projects

None yet

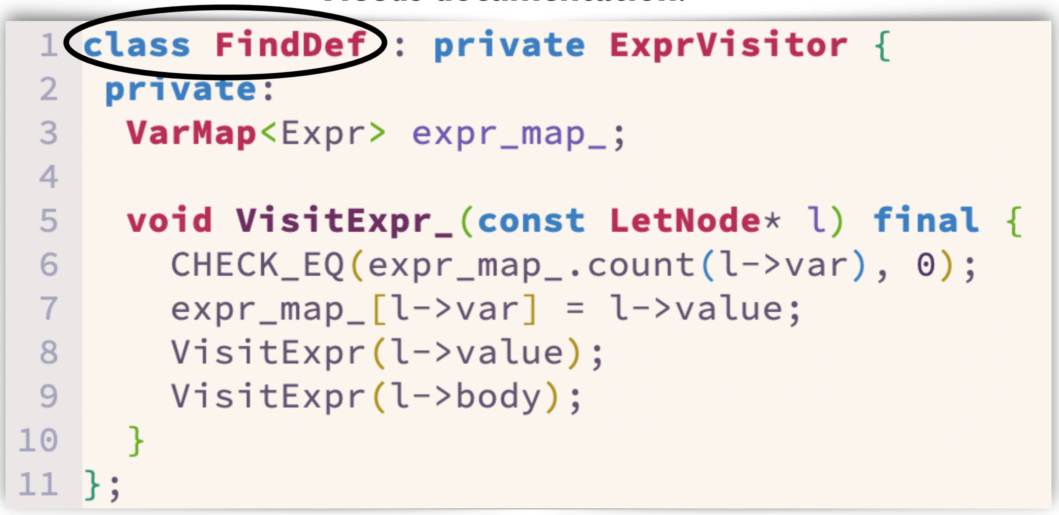


```
class FindDef : private ExprVisitor {
 private:
  VarMap<Expr> expr_map_;

  void VisitExpr_(const LetNode* l) final {
    CHECK_EQ(expr_map_.count(l->var), 0);
    expr_map_[l->var] = l->value;
    VisitExpr(l->value);
    VisitExpr(l->body);
  }
};
```

Let's look at an example analysis which could be improved by an analysis framework.
This analysis finds the values of variables in Relay programs. It is used in the dead code elimination pass, but you could imagine this analysis being useful elsewhere.
However, before it can be useful elsewhere, it has some problems—problems that could be fixed with an analysis framework.
[build] First, the analysis needs documentation. In addition to code comments and better naming, one of the best ways an analysis could document itself is by leaning on the well-documented features of an analysis framework.
[build] Second, the analysis needs a standard data interchange format so that it can compose with other analyses. This format could be standardized by the analysis framework.
[build] Lastly, the analysis needs to be discoverable and accessible to all developers. This could be achieved by having all analyses register themselves with the analysis framework.

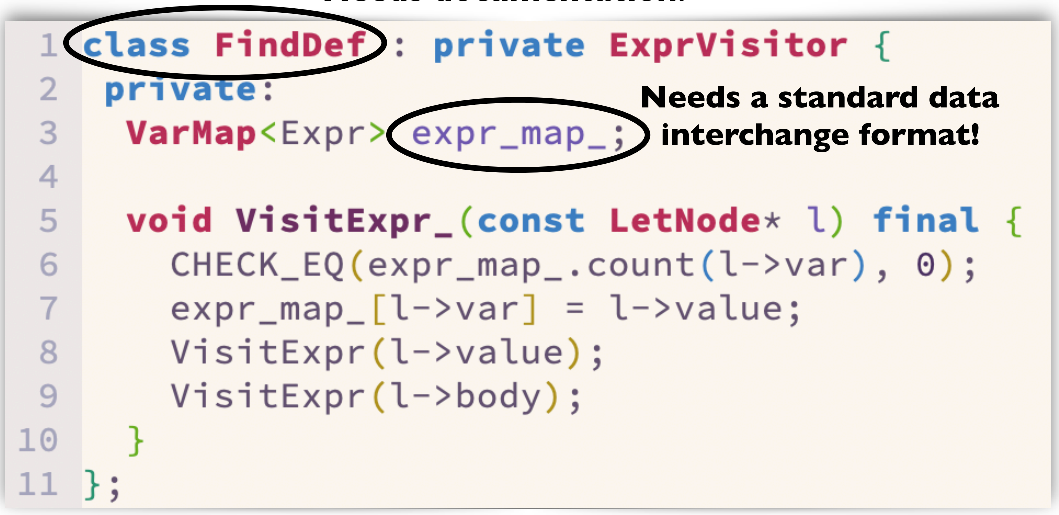
**Needs documentation!**



```
class FindDef : private ExprVisitor {
 private:
  VarMap<Expr> expr_map_;

  void VisitExpr_(const LetNode* l) final {
    CHECK_EQ(expr_map_.count(l->var), 0);
    expr_map_[l->var] = l->value;
    VisitExpr(l->value);
    VisitExpr(l->body);
  }
};
```

**Needs documentation!**

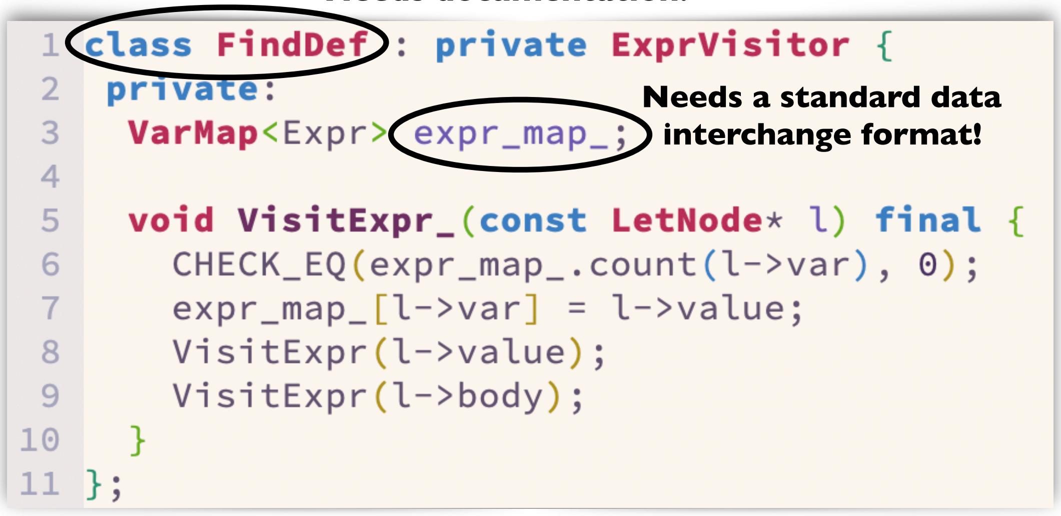


```
class FindDef : private ExprVisitor {
 private:
  VarMap<Expr> expr_map_;

  void VisitExpr_(const LetNode* l) final {
    CHECK_EQ(expr_map_.count(l->var), 0);
    expr_map_[l->var] = l->value;
    VisitExpr(l->value);
    VisitExpr(l->body);
  }
};
```

**Needs a standard data interchange format!**

**Needs documentation!**

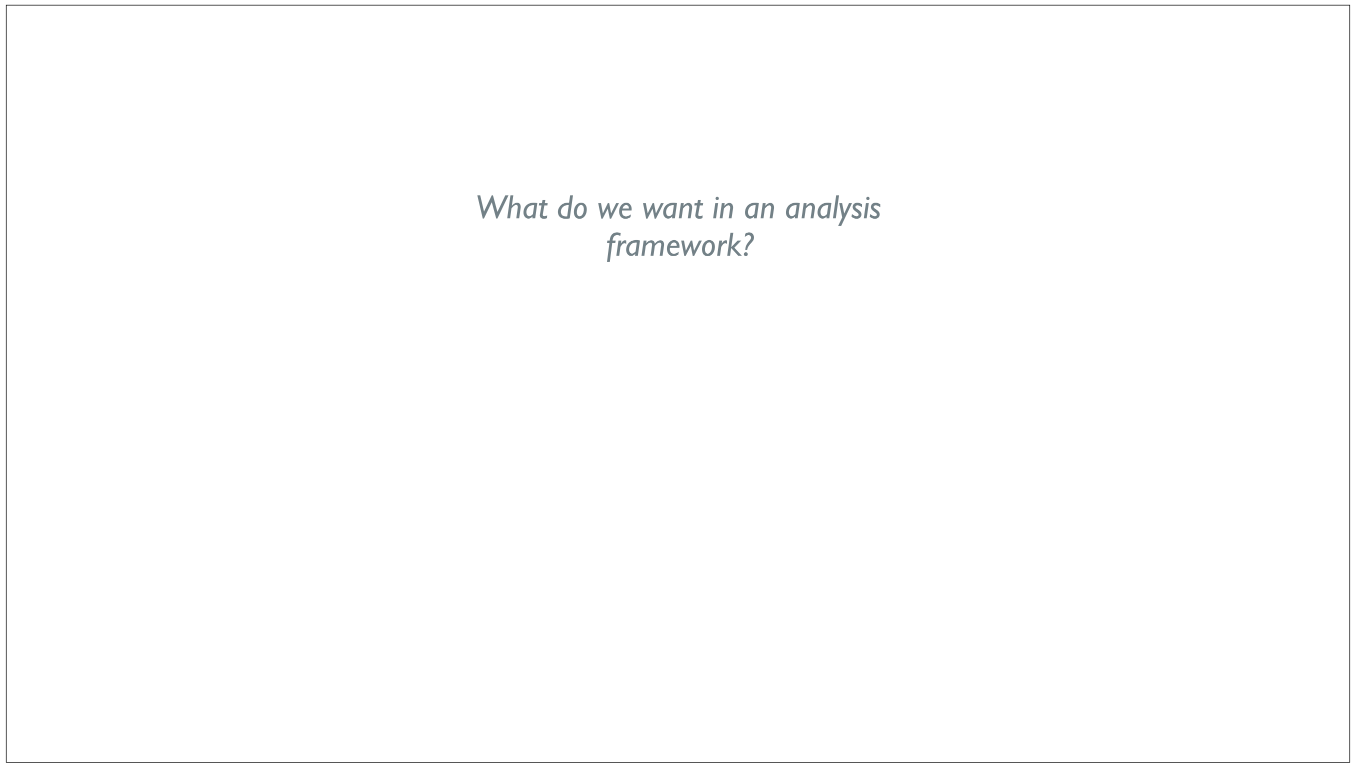


```
class FindDef : private ExprVisitor {
 private:
  VarMap<Expr> expr_map_;

  void VisitExpr_(const LetNode* l) final {
    CHECK_EQ(expr_map_.count(l->var), 0);
    expr_map_[l->var] = l->value;
    VisitExpr(l->value);
    VisitExpr(l->body);
  }
};
```

**Needs a standard data interchange format!**

**...and needs to be discoverable/accessible!**



*What do we want in an analysis framework?*

So, in the end, what we want is an analysis framework with the following features:
[build] The framework should support many types of static and dynamic analyses, and doesn’t limit us on the analyses we can build.
[build] The framework should make it quick and easy to write new analyses, especially for new developers.
[build] And the framework should promote composing analyses together, so that larger analyses can be built off of simpler analyses.

*What do we want in an analysis framework?*

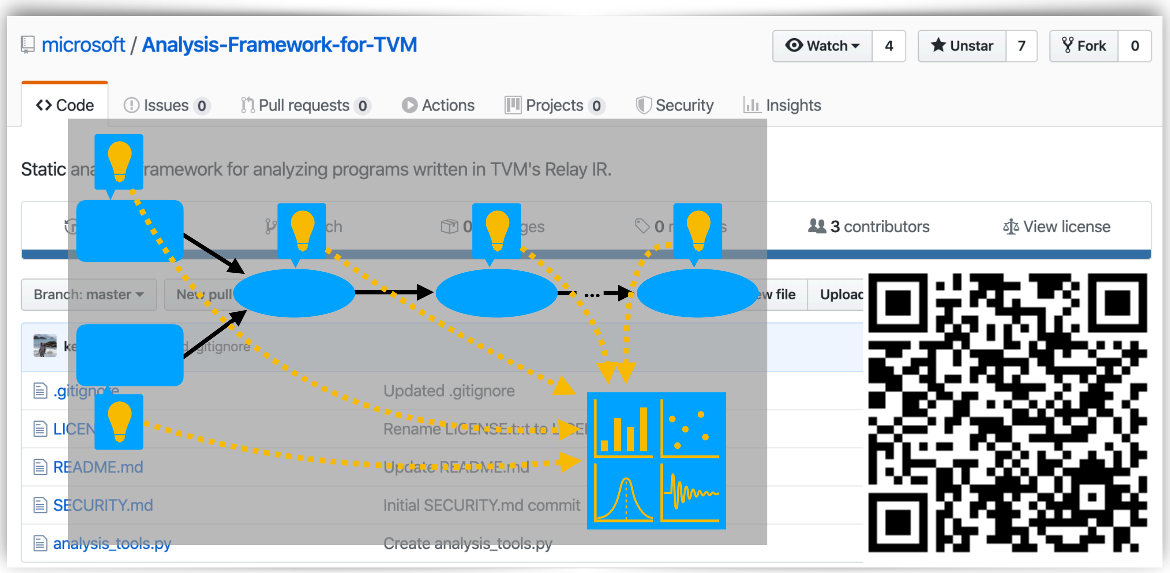
- Supports many types of program analyses

*What do we want in an analysis framework?*

- Supports many types of program analyses
- Quick to write new analyses

*What do we want in an analysis framework?*

- Supports many types of program analyses
- Quick to write new analyses
- Promotes composing analyses together

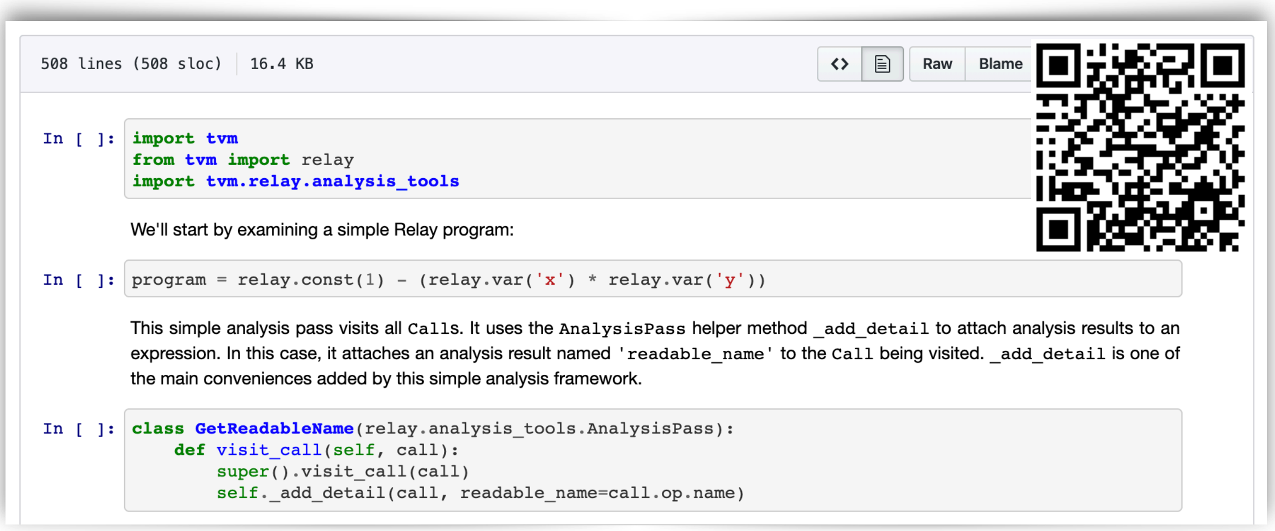

microsoft / Analysis-Framework-for-TVM

Static analysis framework for analyzing programs written in TVM's Relay IR.

| | |
|---|---|
| .gitignore | Updated .gitignore |
| LICENSE | Rename LICENSE.txt to LICENSE |
| README.md | Update README.md |
| SECURITY.md | Initial SECURITY.md commit |
| analysis_tools.py | Create analysis_tools.py |

https://github.com/microsoft/Analysis-Framework-for-TVM

The small program analysis framework I built this past summer at Microsoft achieves some of these goals, at least for simple static analyses. Its code is available on Microsoft's GitHub page, if you want to check it out.


508 lines (508 sloc) | 16.4 KB

```
In [ ]: import tvm
        from tvm import relay
        import tvm.relay.analysis_tools
```

We'll start by examining a simple Relay program:

```
In [ ]: program = relay.const(1) - (relay.var('x') * relay.var('y'))
```

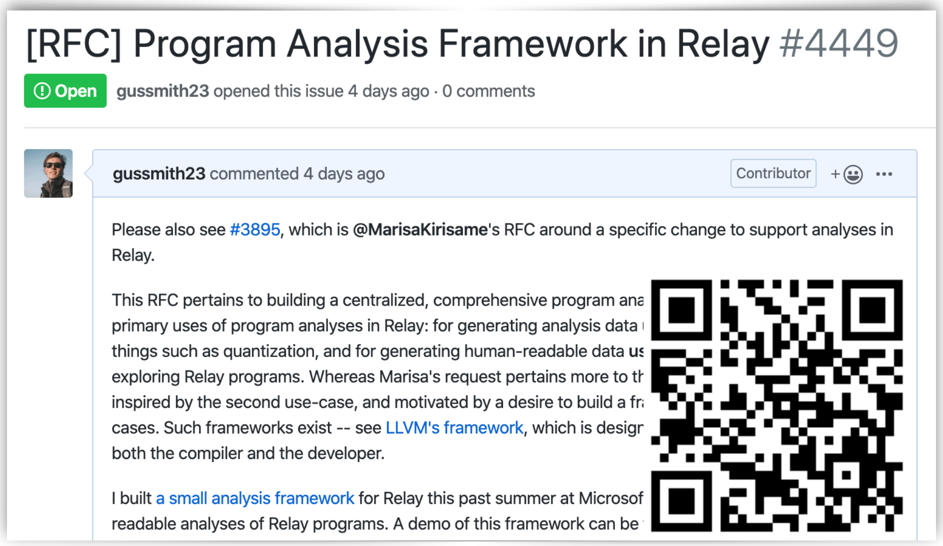
This simple analysis pass visits all `Calls`. It uses the `AnalysisPass` helper method `_add_detail` to attach analysis results to an expression. In this case, it attaches an analysis result named `'readable_name'` to the `Call` being visited. `_add_detail` is one of the main conveniences added by this simple analysis framework.

```
In [ ]: class GetReadableName(relay.analysis_tools.AnalysisPass):
            def visit_call(self, call):
                super().visit_call(call)
                self._add_detail(call, readable_name=call.op.name)
```

https://github.com/gussmith23/tvm/blob/analysis-framework-demo/demo.ipynb

In addition, I have a demo using the analysis framework on my fork of TVM.

## Moving forward


[RFC] Program Analysis Framework in Relay #4449

Open gussmith23 opened this issue 4 days ago · 0 comments

gussmith23 commented 4 days ago Contributor

Please also see #3895, which is @MarisaKirisame's RFC around a specific change to support analyses in Relay.

This RFC pertains to building a centralized, comprehensive program ana
primary uses of program analyses in Relay: for generating analysis data
things such as quantization, and for generating human-readable data us
exploring Relay programs. Whereas Marisa's request pertains more to th
inspired by the second use-case, and motivated by a desire to build a fr
cases. Such frameworks exist -- see LLVM's framework, which is design
both the compiler and the developer.

I built a small analysis framework for Relay this past summer at Microsof
readable analyses of Relay programs. A demo of this framework can be

https://github.com/apache/incubator-tvm/issues/4449

The whole point of this lightning talk is to show that there is need and desire for analysis tooling in Relay.
If you have any opinions on a potential program analysis framework, please contribute to the RFC I opened on the TVM GitHub.
This is not my primary research, and I do not plan on building the analysis framework myself, but I'm happy to help organize and encourage the discussion!

Thanks everybody!