Identifying pass conflicts in LLVM

Sébastien Michelland (ENS Lyon)

Sébastien Mosser (UQÀM)
Laure Gonnord, Matthieu Moy (Université Lyon 1)

Second meeting of the SE@MTL community

June 6, 2019

There is hidden complexity in compilation pass order

```
% clang -O3 -emit-llvm program.c -o program.bc
```

- 260 passes using with 61 optimizations and 24 analyses

```
... -domtree -loops -loop-simplify ...
```

- Order of optimizations counts!
- Only constrained by dependencies specified by humans.

Can we optimize further by reordering passes?

The LLVM test suite provides experimental proof

- Modified to run arbitrary tests on a wide set of programs

```
% opt -loop-simplify -loop-simplify program.bc
```

- Want a mathematical model, but...

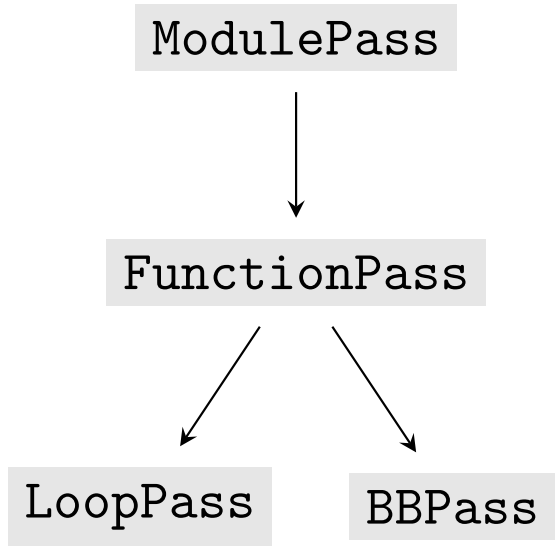
$$(P \bullet x) \bullet y \quad \text{should be} \quad P \bullet (x \ ; \ y)$$

`opt -x P | opt -y` is not really `opt -x -y P`

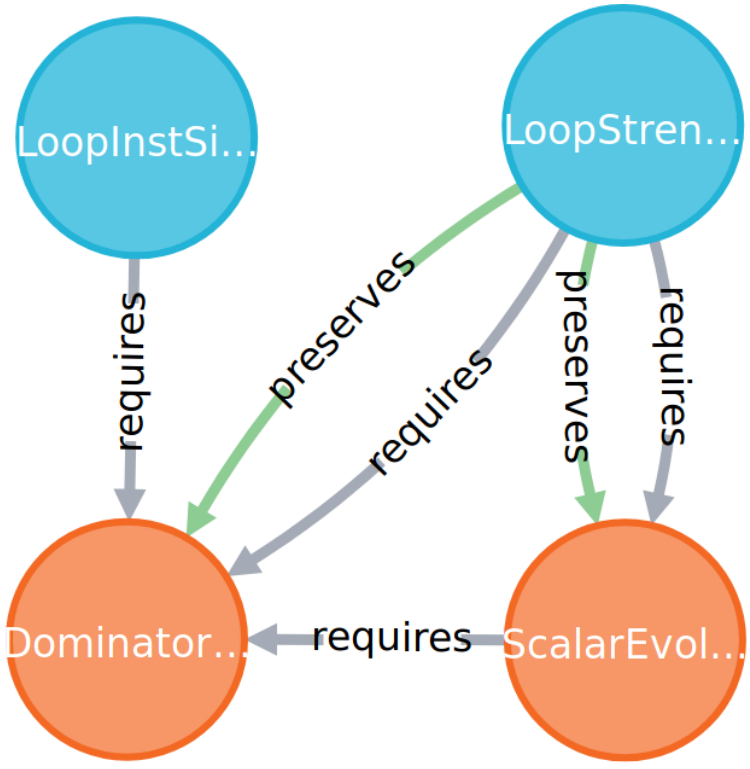
- Engineering challenge.

Not all orderings are interesting

- Find passes that don't commute... and are semantically related.
- Typing: which areas of the program are affected
- Common dependencies and preserved analyzes
- Need to visualize the types and relations.



An example of pass relations



- Automated extraction from LLVM source code (227 passes, 501 relations)
- Neo4j database
- Almost all optimizations are independent from each other

Underlying research questions

- Methodology to locate interesting pairs?
  - Pairwise testing
- Which notion of equivalent orders?
- ... which notion of equivalent programs?
  - Program semantics are preserved by LLVM
  - Code equality, structural equality, speed
- Is the dependency information reliable?