Taking Back Control: Formally Modelling a Compiler Intermediate Representation for GPU Computing

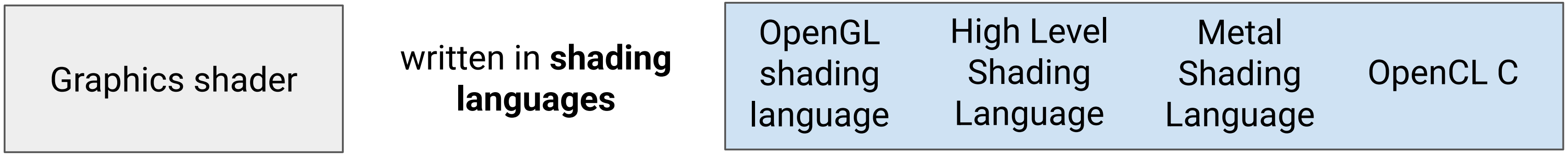
Alastair F. Donaldson, Imperial College London, UK
Joint work with Vasileos Klimis, Jack Clark and John Wickerson (Imperial), Alan Baker and David Neto (Google)

Verification Seminar Series, March 2025

Overview

- Motivation for SPIR-V
- Outline of approach to improving SPIR-V
- Problems with SPIR-V definitions
- Our solution: structural dominance
- Bonus: a new method for compiler fuzzing

Graphics shaders



Graphics shader

written in **shading languages**

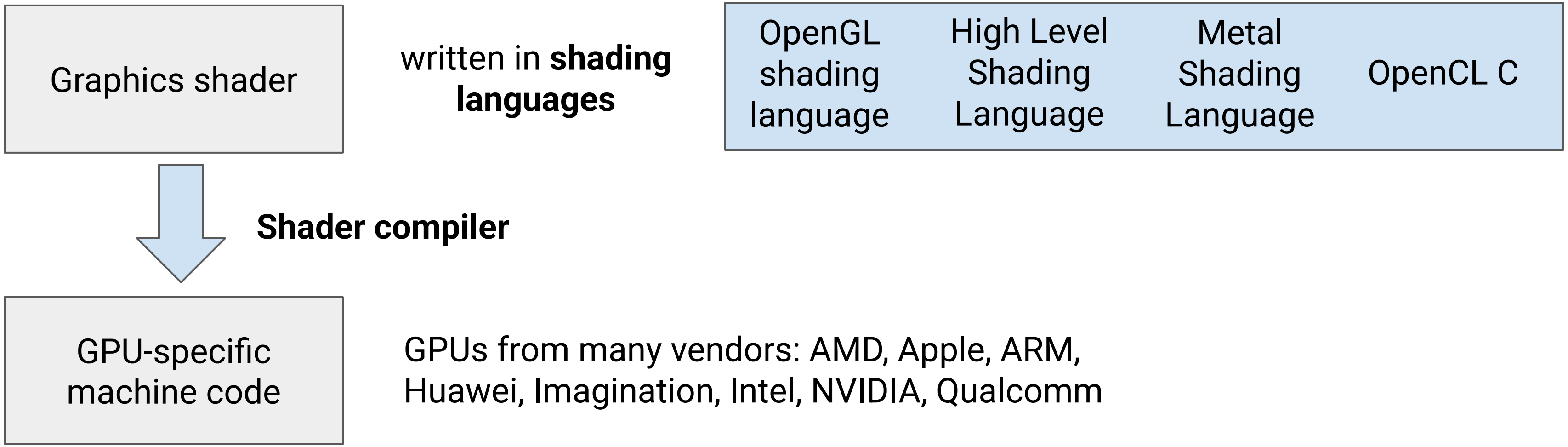
OpenGL shading language

High Level Shading Language

Metal Shading Language

OpenCL C

Graphics shaders



Graphics shader

written in **shading languages**

OpenGL shading language

High Level Shading Language

Metal Shading Language

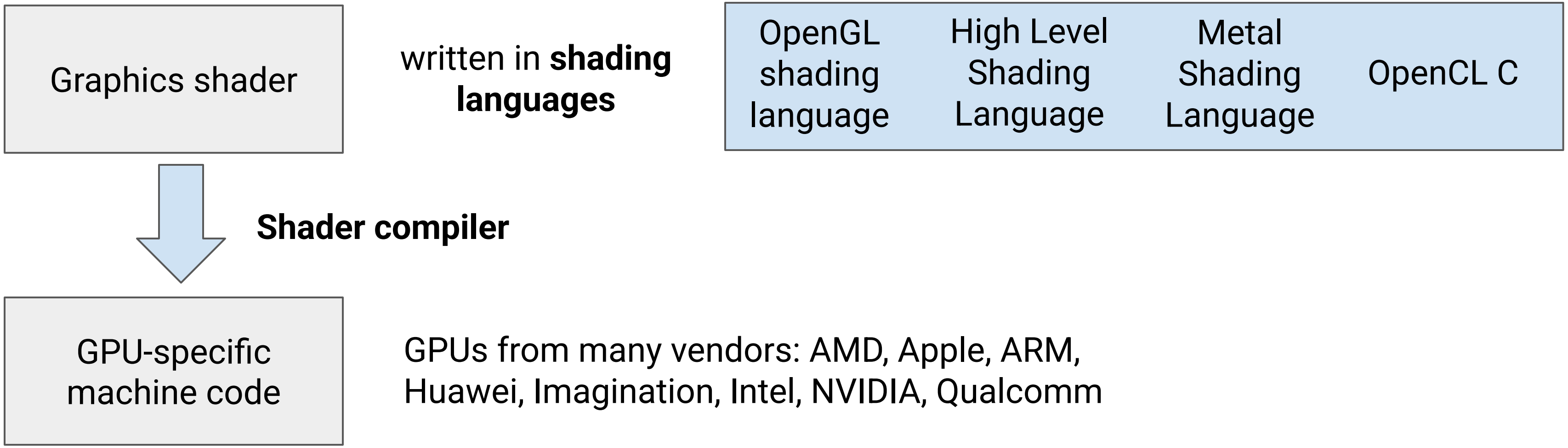
OpenCL C

Shader compiler

GPU-specific machine code

GPUs from many vendors: AMD, Apple, ARM, Huawei, Imagination, Intel, NVIDIA, Qualcomm

Graphics shaders



Graphics shader

written in **shading languages**

OpenGL shading language

High Level Shading Language

Metal Shading Language

OpenCL C

Shader compiler

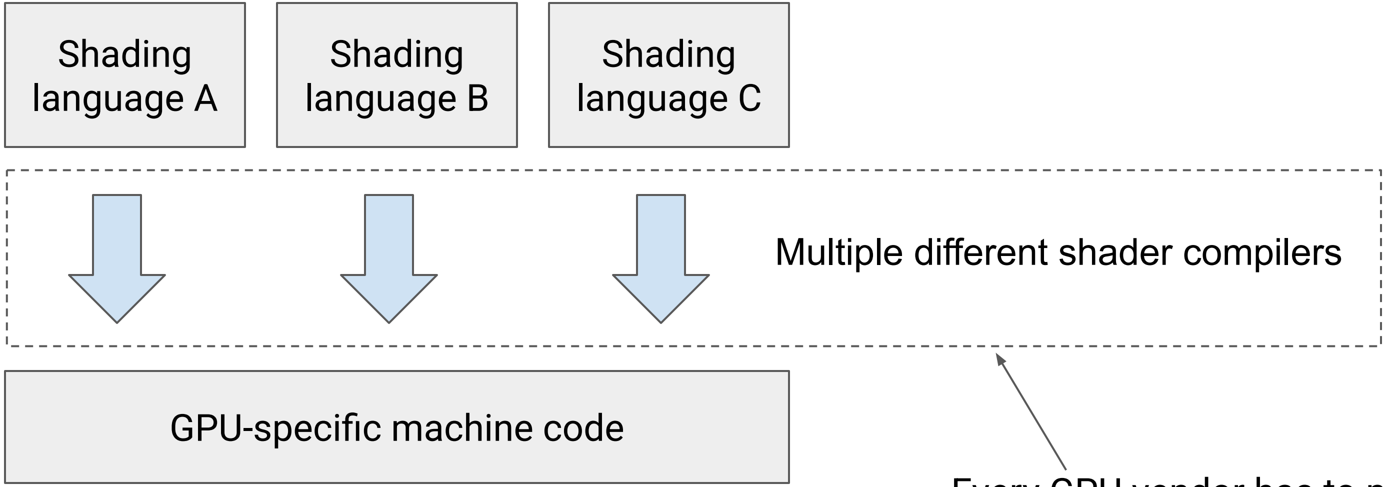
GPU-specific machine code

GPUs from many vendors: AMD, Apple, ARM, Huawei, Imagination, Intel, NVIDIA, Qualcomm

Shader compiler: the **most complex** part of a GPU device driver

SPIR-V: Standard, Portable Intermediate Representation

Motivation



Shading language A

Shading language B

Shading language C

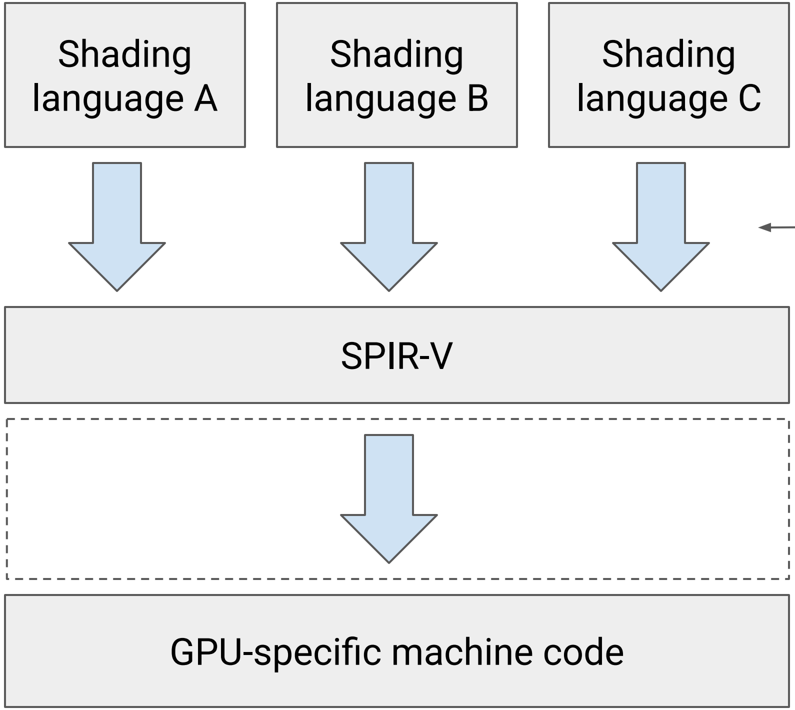
Multiple different shader compilers

GPU-specific machine code

Every GPU vendor has to maintain their own set of shader compilers: a lot of work

SPIR-V: Standard, Portable Intermediate Representation

Motivation



Industry standard, GPU-agnostic translators

Every GPU vendor writes a compiler for SPIR-V - reduces overall burden

SPIR-V specification had some major problems

Problems related to sophisticated rules about control flow

Intended to help developers and compiler writers

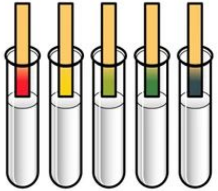
Not helping in practice:

- Dzmitry Malyshau, Mozilla: Horrors of SPIR-V
- Sean Baxter, Circle compiler: Targeting SPIR-V is super easy and the structurization requirements totally won't make you want to throw yourself off a cliff
- Hans-Kristian Arntzen, Arntzen Software: My personal hell of translating DXIL to SPIR-V

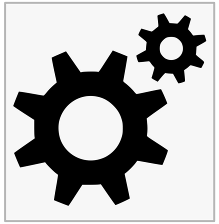
Sources of truth about SPIR-V

Prose specification

Conformance test suites



Validation tooling



Experts

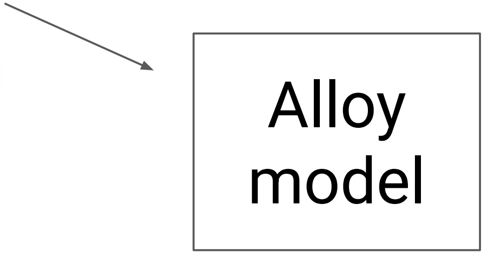
David

Alan

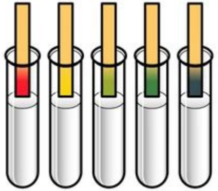
Modelling SPIR-V control flow in Alloy

Prose specification

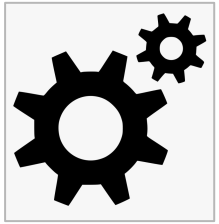
Best-effort initial interpretation



Conformance test suites



Validation tooling



Experts

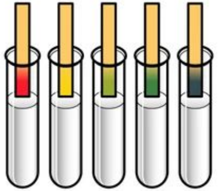
David

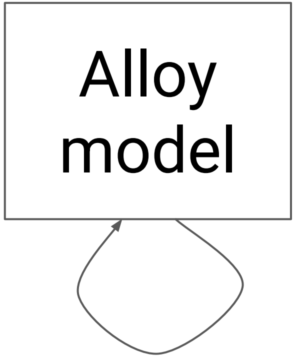
Alan

Modelling SPIR-V control flow in Alloy

Prose specification

Conformance test suites

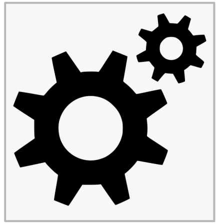




Alloy model

Formulate solutions to known problems

Validation tooling



Experts

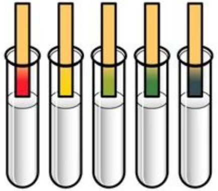
David

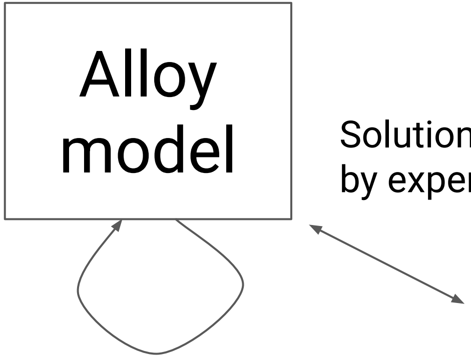
Alan

Modelling SPIR-V control flow in Alloy

Prose specification

Conformance test suites



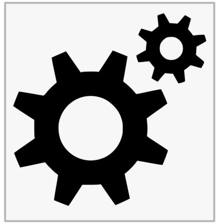


Alloy model

Solutions informed by experts

Formulate solutions to known problems

Validation tooling



Experts

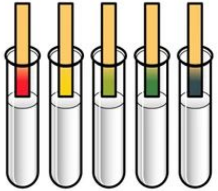
David

Alan

Modelling SPIR-V control flow in Alloy

Prose specification

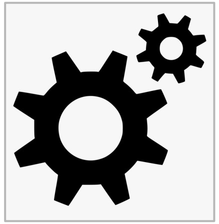
Conformance test suites



Cross-check against test suites

Alloy model

Validation tooling



Experts

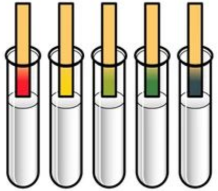
David

Alan

Modelling SPIR-V control flow in Alloy

Prose specification

Conformance test suites



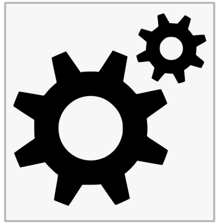
Cross-check
against test suites

Alloy
model

Fix ill-formed tests

Consult with
experts

Validation tooling

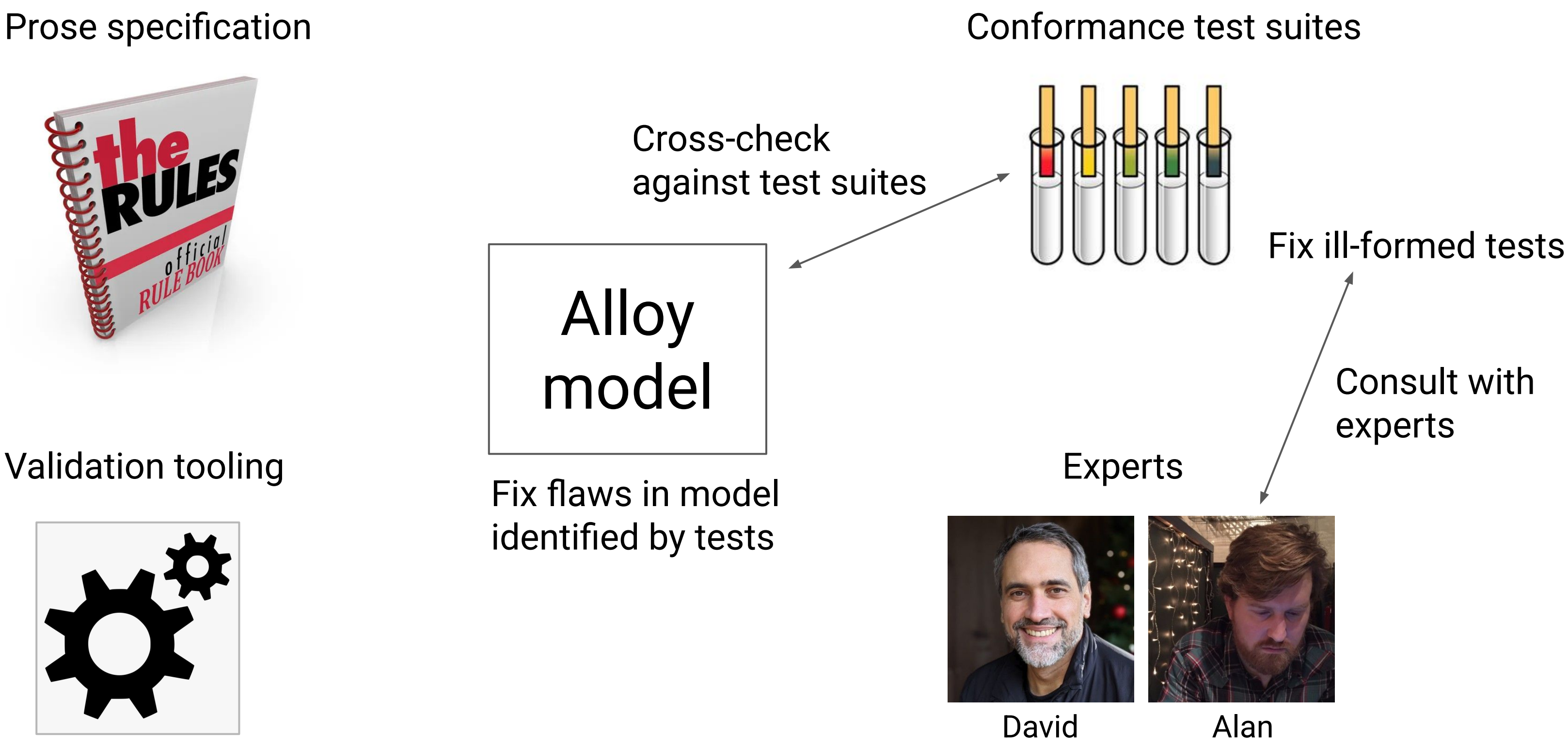


Experts

David

Alan

Modelling SPIR-V control flow in Alloy



Prose specification

Validation tooling

Alloy model

Fix flaws in model identified by tests

Cross-check against test suites

Conformance test suites

Fix ill-formed tests

Consult with experts

Experts

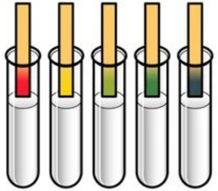
David

Alan

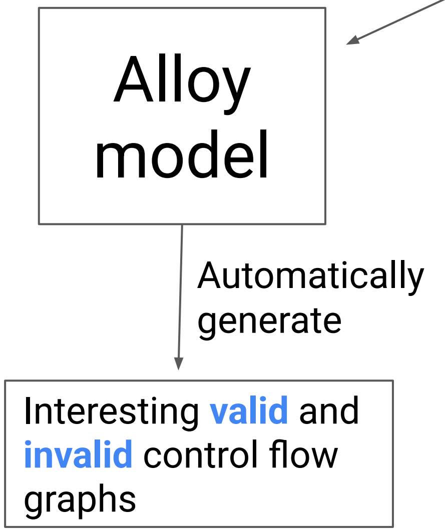
Modelling SPIR-V control flow in Alloy

Prose specification

Conformance test suites



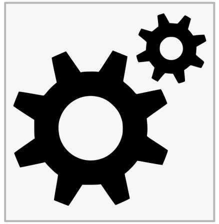
Agreement



Automatically generate

Interesting **valid** and **invalid** control flow graphs

Validation tooling

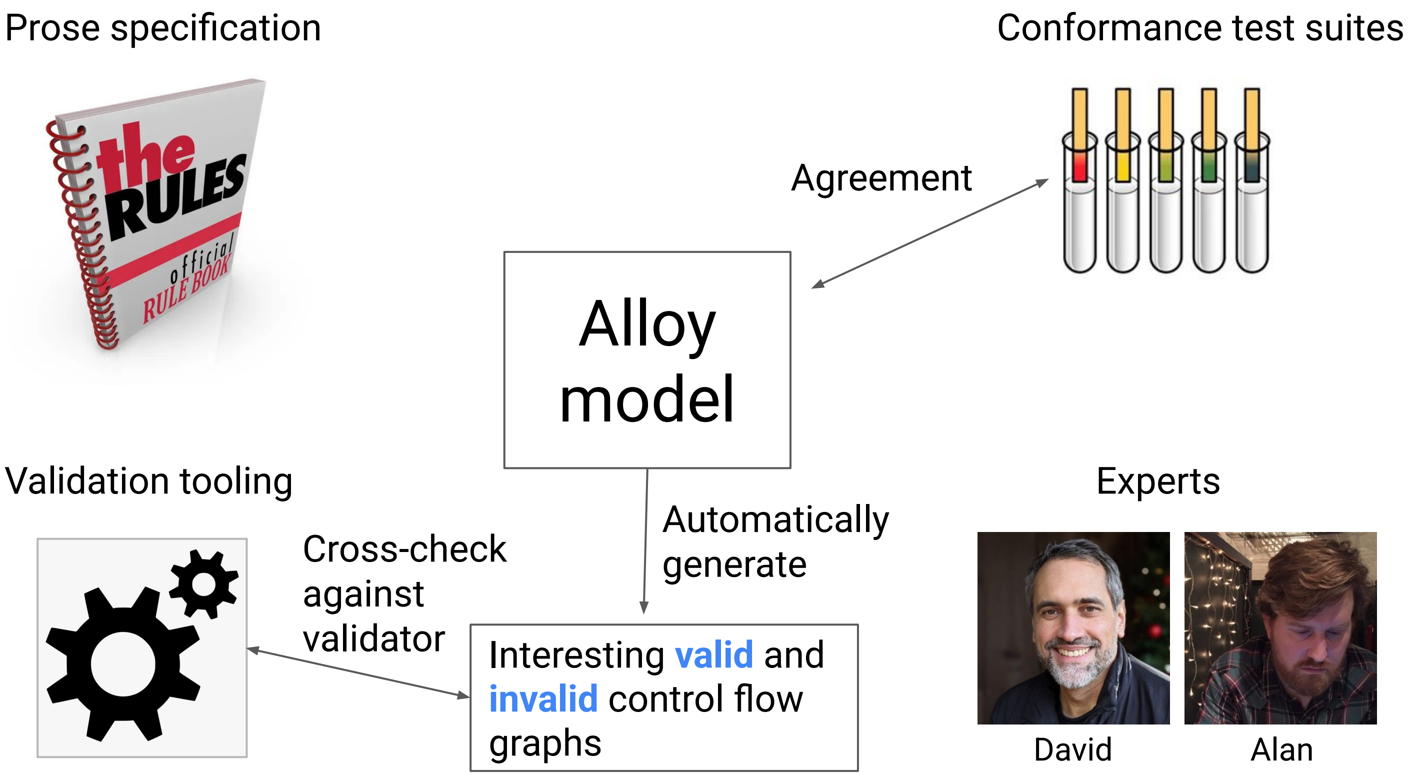


Experts

David

Alan

Modelling SPIR-V control flow in Alloy



Prose specification

Conformance test suites

Agreement

Alloy model

Automatically generate

Interesting **valid** and **invalid** control flow graphs

Cross-check against validator

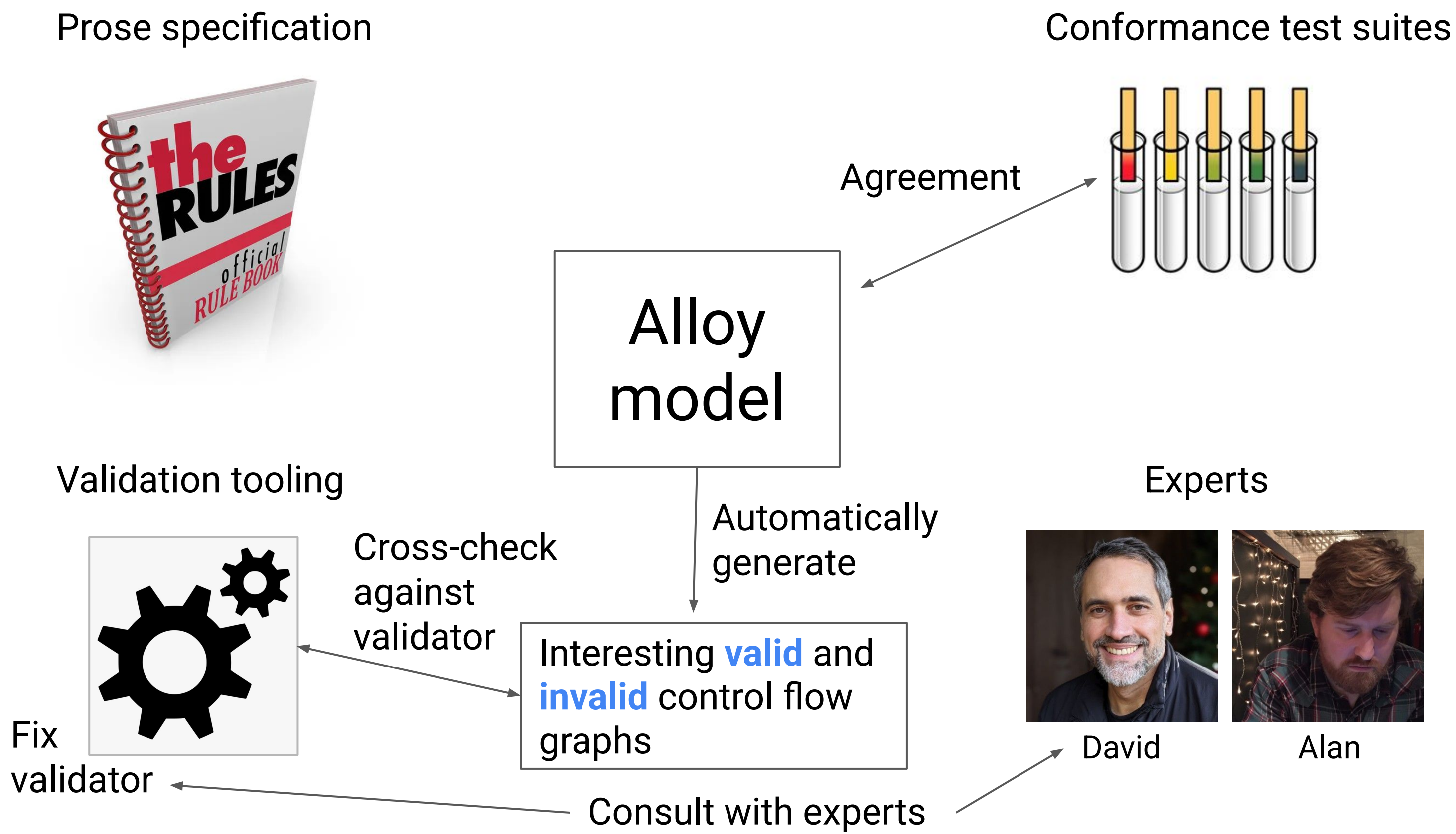
Validation tooling

Experts

David

Alan

Modelling SPIR-V control flow in Alloy



Prose specification

Conformance test suites

Agreement

Alloy model

Automatically generate

Validation tooling

Cross-check against validator

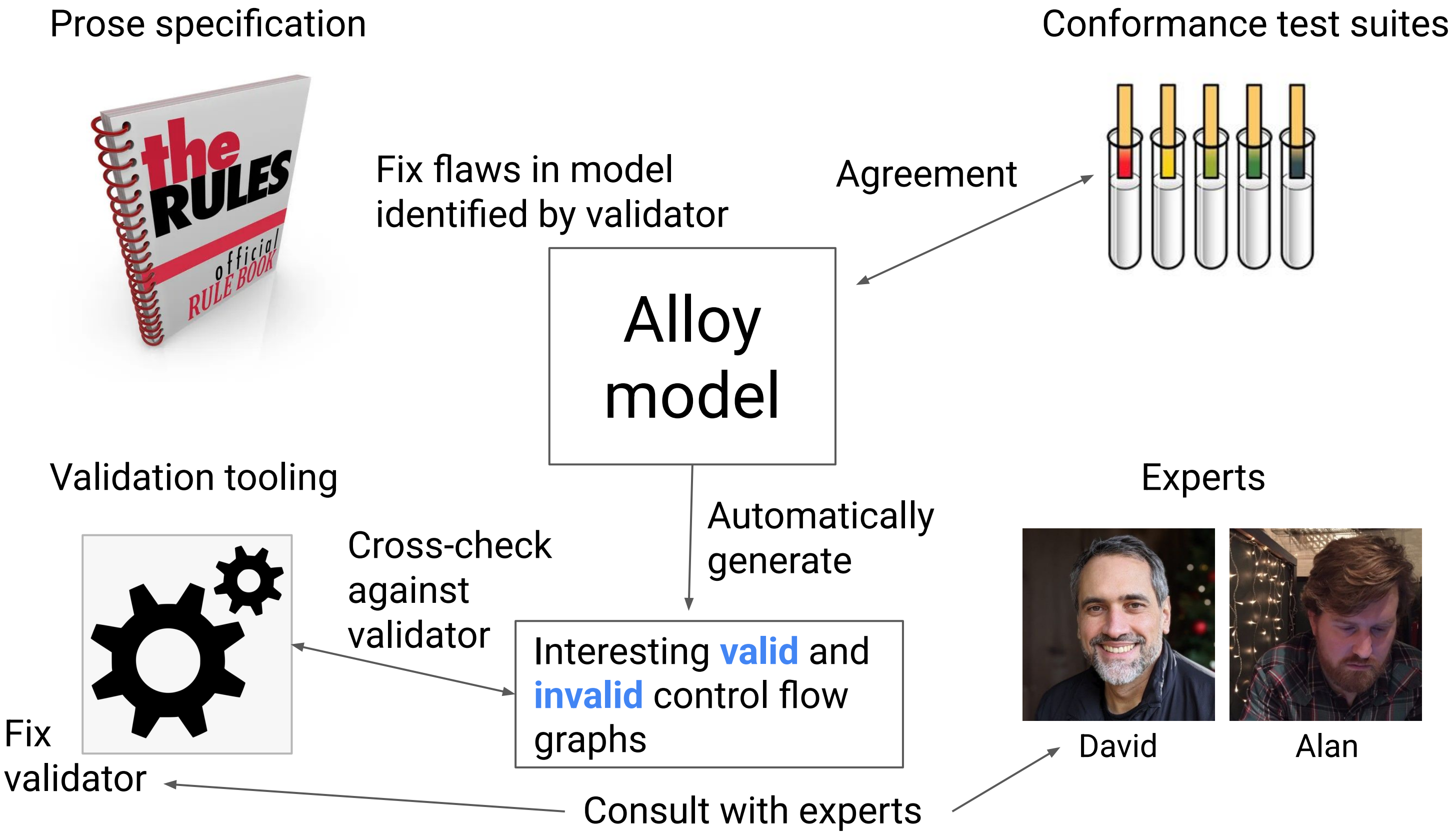
Interesting **valid** and **invalid** control flow graphs

Experts

Fix validator

Consult with experts

Modelling SPIR-V control flow in Alloy



Prose specification

Fix flaws in model identified by validator

Agreement

Conformance test suites

Alloy model

Validation tooling

Automatically generate

Experts

Cross-check against validator

Interesting valid and invalid control flow graphs

Fix validator

Consult with experts

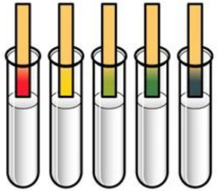
David

Alan

Modelling SPIR-V control flow in Alloy

Prose specification

Conformance test suites



Fix flaws in model identified by validator

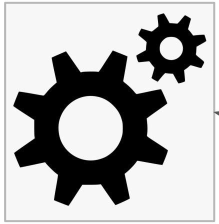
Cross-check against test suites

Alloy model

Automatically generate

Interesting valid and invalid control flow graphs

Validation tooling



Cross-check against validator

Fix validator

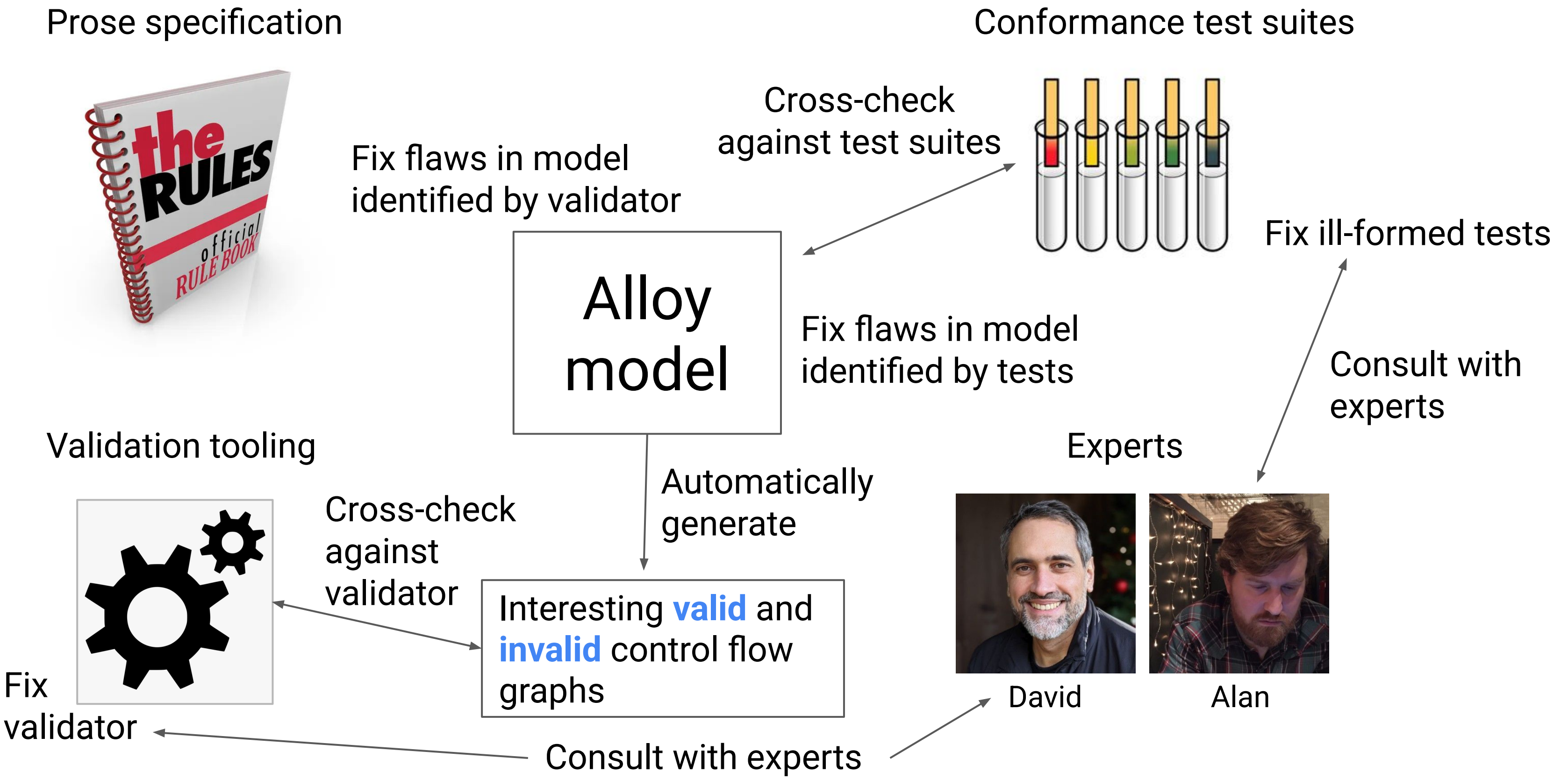
Experts

David

Alan

Consult with experts

Modelling SPIR-V control flow in Alloy



Prose specification

Conformance test suites

Fix flaws in model identified by validator

Cross-check against test suites

Alloy model

Fix ill-formed tests

Fix flaws in model identified by tests

Consult with experts

Validation tooling

Experts

Automatically generate

Cross-check against validator

Interesting **valid** and **invalid** control flow graphs

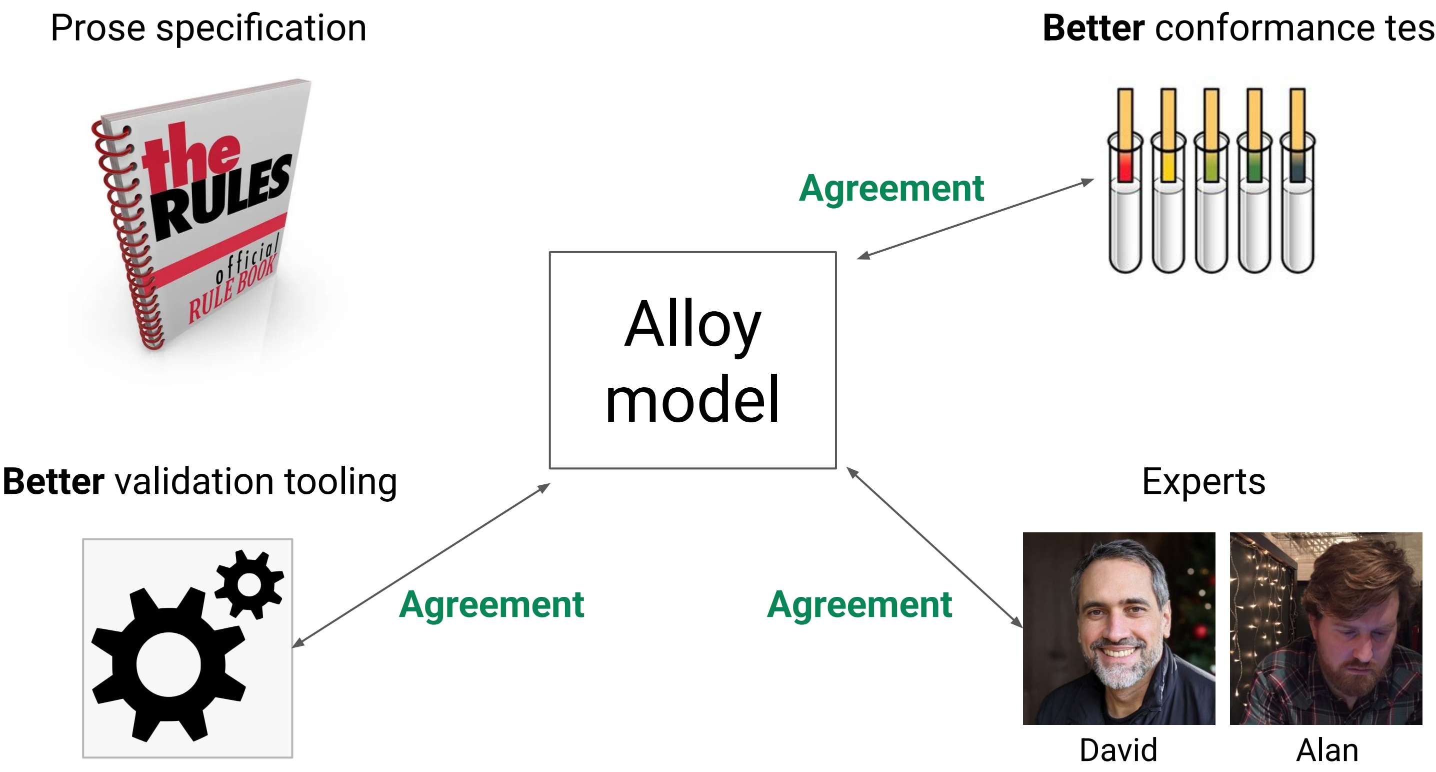
David

Alan

Fix validator

Consult with experts

Virtuous cycle improved formal model, conformance tests + tooling



Prose specification

Better conformance test suites

Agreement

Alloy model

Better validation tooling

Agreement

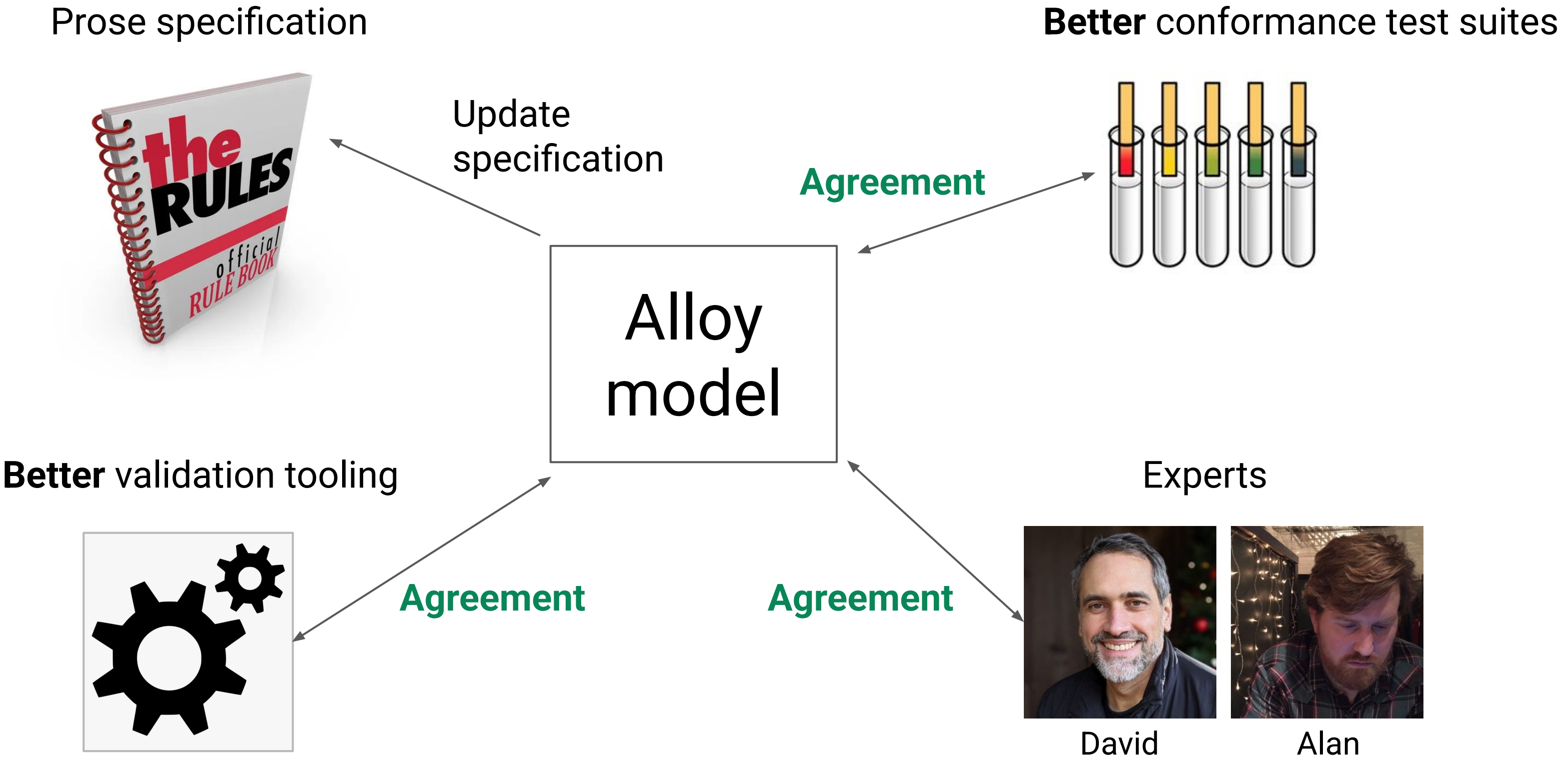
Agreement

Experts

David

Alan

Virtuous cycle improved formal model, conformance tests + tooling



Prose specification

Better conformance test suites

Update specification

Agreement

Alloy model

Better validation tooling

Experts

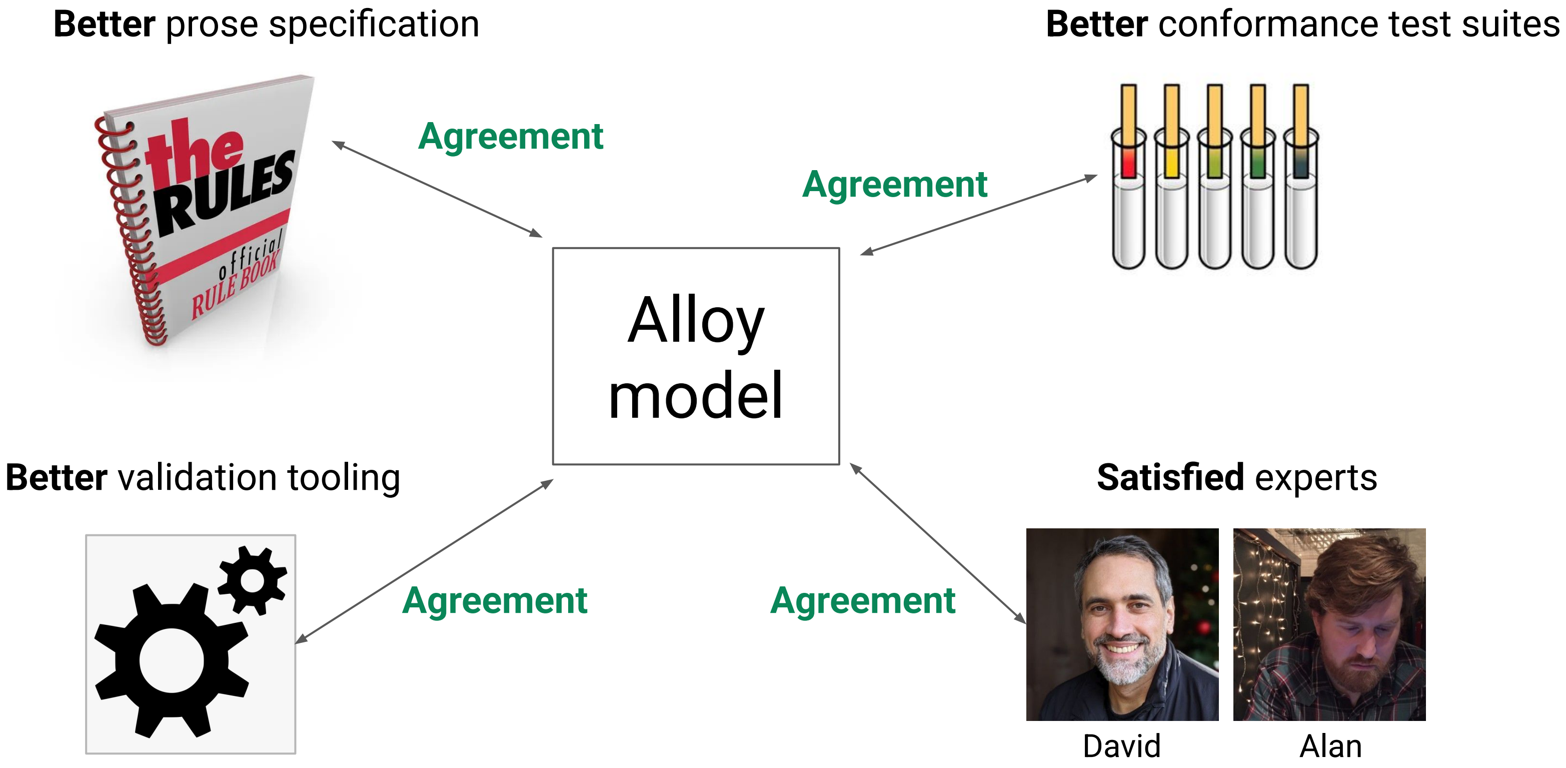
Agreement

Agreement

David

Alan

Our changes are now integrated into the SPIR-V specification



Better prose specification

Agreement

Better conformance test suites

Agreement

Alloy model

Better validation tooling

Agreement

Agreement

Satisfied experts

David

Alan

Insight into problems with SPIR-V control flow

Let’s look at the definitions of control flow constructs in SPIR-V before our changes

Structured control flow in SPIR-V

High level program

```
void main() {
  int x = 0, i = 0; // %1
  while(i < 100) { // %2
    if (i < 50) // %3
      x += 2; // %4
    else
      x += 4; // %5
    // %6
    i++; // %7
  } // %8
}
```

Possible SPIR-V representation

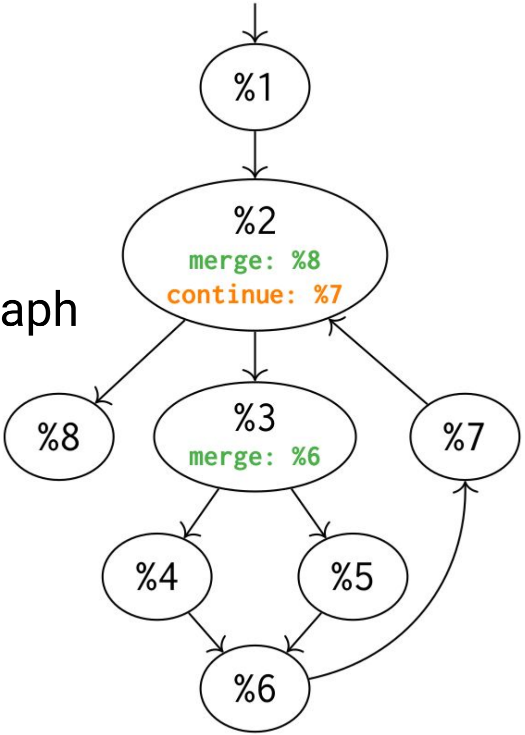
```
%44 = OpFunction %42 None %43
%1  = OpLabel
%48 = OpVariable %47 Function
%10 = OpVariable %47 Function
      OpStore %48 %9
      OpStore %10 %9
      OpBranch %2
%2  = OpLabel
%16 = OpLoad %46 %10
%19 = OpSLessThan %18 %16 %17
      OpLoopMerge %8 %7 None
      OpBranchConditional %19 %3 %8
%3  = OpLabel
%20 = OpLoad %46 %10
%22 = OpSLessThan %18 %20 %21
      OpSelectionMerge %6 None
      OpBranchConditional %22 %4 %5
%4  = OpLabel
%26 = OpLoad %46 %48
%27 = OpIAdd %46 %26 %25
      OpStore %48 %27
      OpBranch %6
%5  = OpLabel
%30 = OpLoad %46 %48
%31 = OpIAdd %46 %30 %29
      OpStore %48 %31
      OpBranch %6
%6  = OpLabel
      OpBranch %7
%7  = OpLabel
%32 = OpLoad %46 %10
%34 = OpIAdd %46 %32 %33
      OpStore %10 %34
      OpBranch %2
%8  = OpLabel
      OpReturn
      OpFunctionEnd
```

Structured control flow in SPIR-V

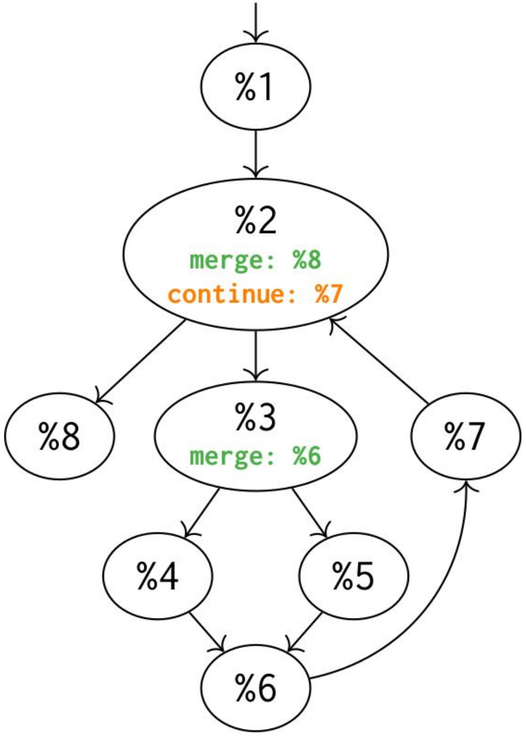
SPIR-V program

```
%44 = OpFunction %42 None %43
%1  = OpLabel
%48 = OpVariable %47 Function
%10 = OpVariable %47 Function
      OpStore %48 %9
      OpStore %10 %9
      OpBranch %2
%2  = OpLabel
%16 = OpLoad %46 %10
%19 = OpSLessThan %18 %16 %17
      OpLoopMerge %8 %7 None
      OpBranchConditional %19 %3 %8
%3  = OpLabel
%20 = OpLoad %46 %10
%22 = OpSLessThan %18 %20 %21
      OpSelectionMerge %6 None
      OpBranchConditional %22 %4 %5
%4  = OpLabel
%26 = OpLoad %46 %48
%27 = OpIAdd %46 %26 %25
      OpStore %48 %27
      OpBranch %6
%5  = OpLabel
%30 = OpLoad %46 %48
%31 = OpIAdd %46 %30 %29
      OpStore %48 %31
      OpBranch %6
%6  = OpLabel
      OpBranch %7
%7  = OpLabel
%32 = OpLoad %46 %10
%34 = OpIAdd %46 %32 %33
      OpStore %10 %34
      OpBranch %2
%8  = OpLabel
      OpReturn
      OpFunctionEnd
```

Control flow graph



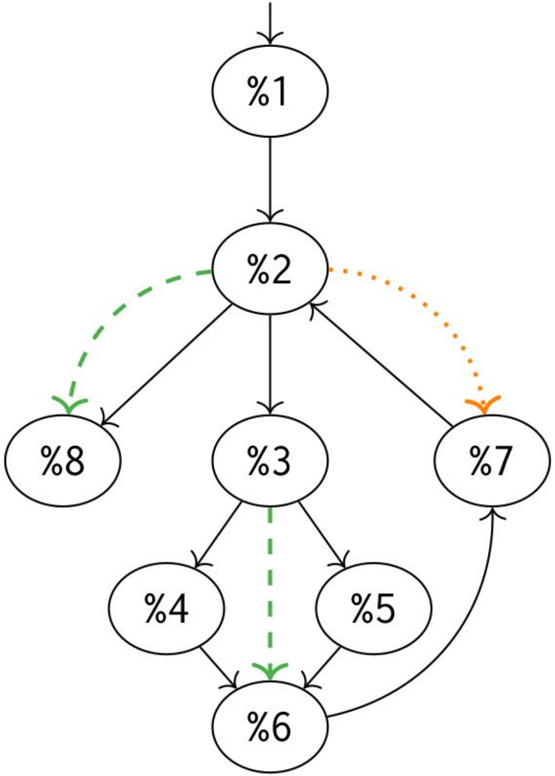
Structured control flow in SPIR-V



Use special edges to record

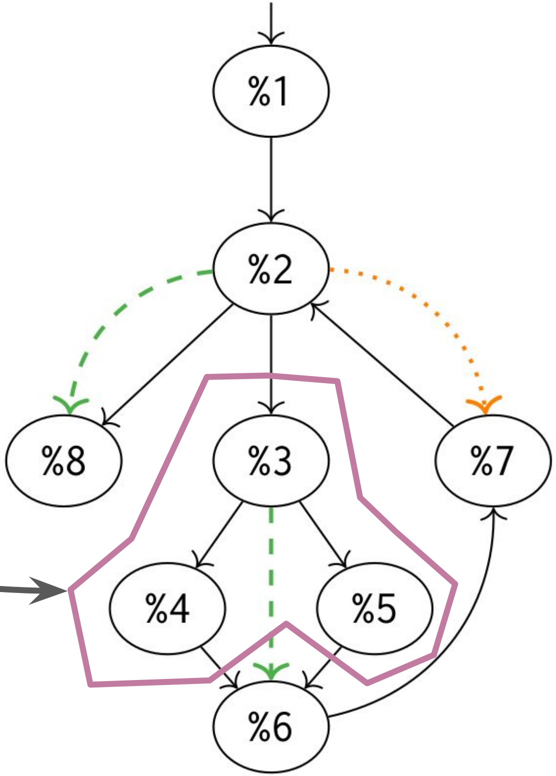
merge blocks – – – – – ➤

continue targets · · · · · · · · ➤



Selection construct: intuitively, the body of an if-then-else

```
void main() {
  int x = 0, i = 0; // %1
  while(i < 100) { // %2
    if (i < 50) // %3
      x += 2; // %4
    else
      x += 4; // %5
    // %6
    i++; // %7
  } // %8
}
```



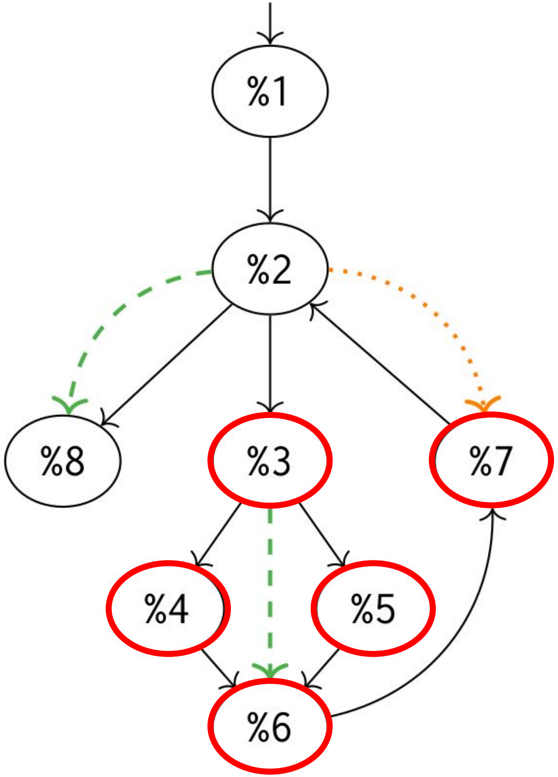
Selection construct: original definition

Path: just ⟶ edges

Block A **dominates** block B if every path from entry to B includes A

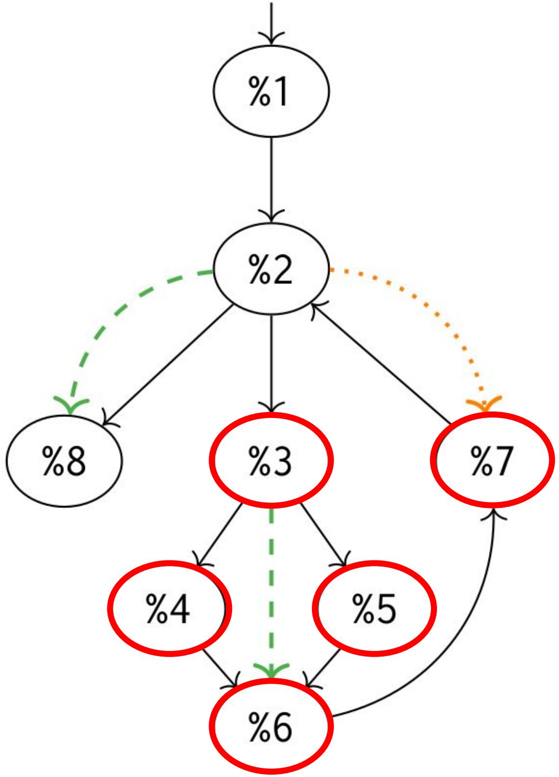
Blocks dominated by 3?

{3, 4, 5, 6, 7}



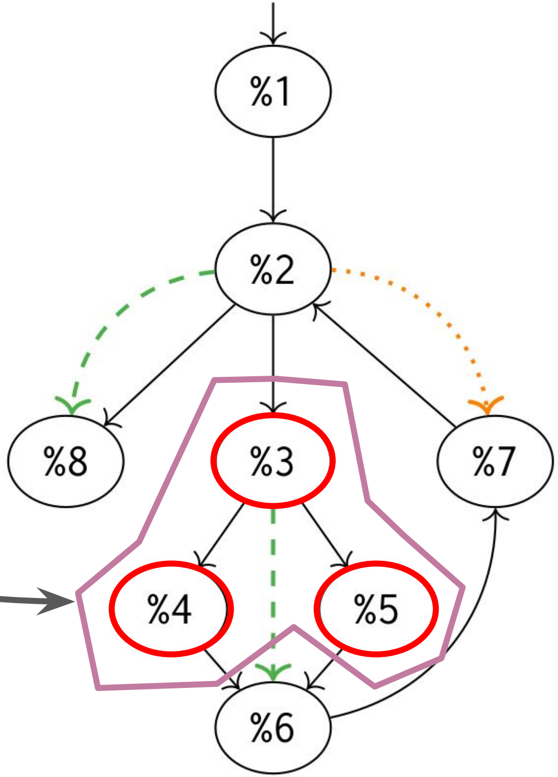
Selection construct: original definition

A **selection construct**: includes the blocks dominated by a selection header, while excluding blocks dominated by the selection construct's merge block



Selection construct: original definition

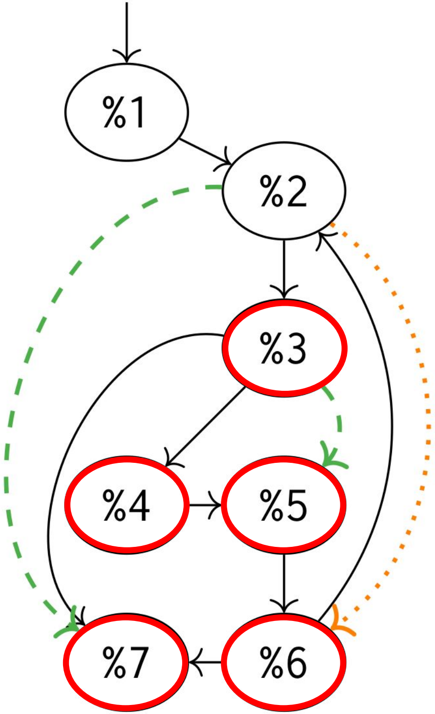
```
void main() {
  int x = 0, i = 0; // %1
  while(i < 100) { // %2
    if (i < 50) // %3
      x += 2; // %4
    else
      x += 4; // %5
    // %6
    i++; // %7
  } // %8
}
```



Works as desired in this example!

Problematic example: loop with early break

```
void main() {
   int i = 0; // %1
   do { // %2
      if (i > 50) { // %3
         break;
      } else {
         // no-op // %4
      }
      i++; // %5
   }
   while(i < 100); // %6
   // %7
}
```



Selection construct headed at 3?

Blocks dominated by 3

{3, 4, 5, 6, 7}

Minus blocks dominated by 3’s merge - i.e. by 5

{5, 6}

Yields {3, 4, 7}

Flawed attempt at a fix

A **selection construct**: includes the blocks dominated by a selection header, while excluding blocks dominated by the selection construct's merge block

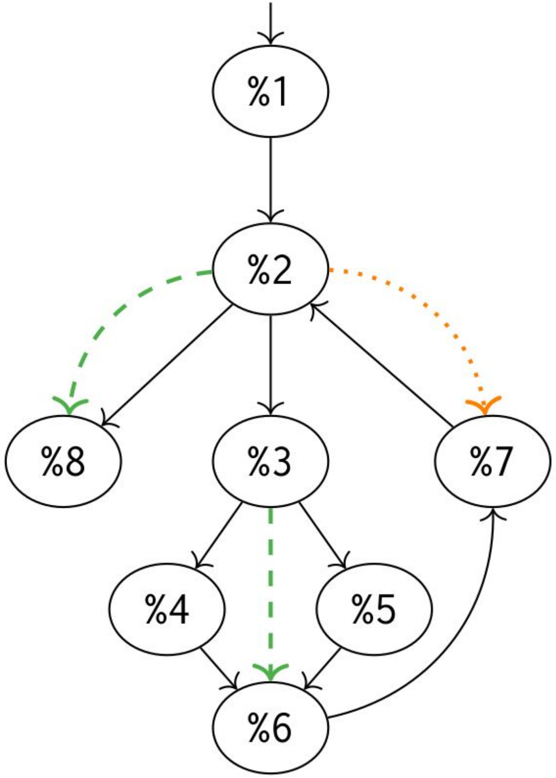
Furthermore, these structured control-flow constructs are additionally defined to exclude all outer constructs' continue constructs and exclude all blocks dominated by all outer constructs' merge blocks.

Complex and circular

Our solution: *structural dominance*

Path: just → edges

Structural path: combination of → ⇢ and ⤑ edges



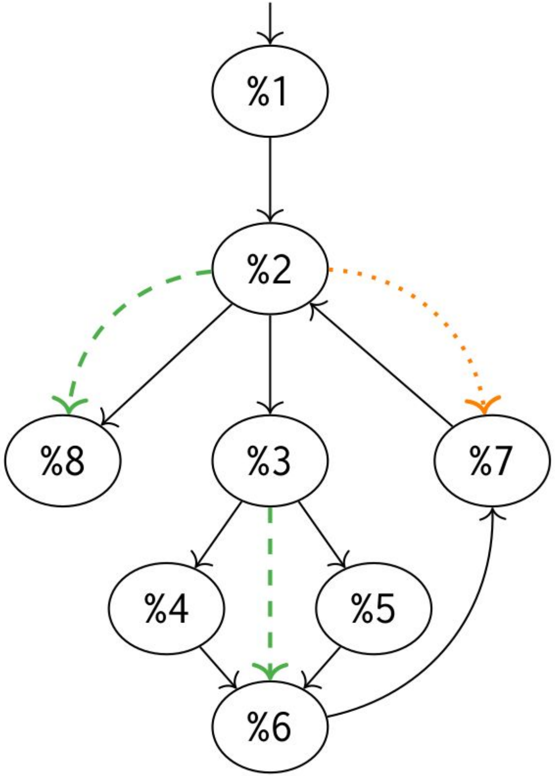
Our solution: *structural dominance*

Block A **dominates** block B if every path from entry to B includes A

Block A **structurally dominates** block B if every structural path from entry to B includes A

Dominated by 3? {3, 4, 5, 6, 7}

Structurally dominated by 3? {3, 4, 5, 6}



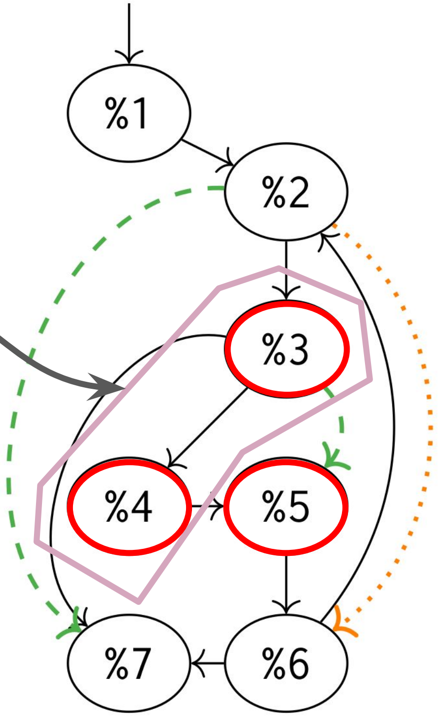
Selection construct revisited

A **selection construct**: includes the blocks structurally dominated by a selection header, while excluding blocks structurally dominated by the selection construct’s merge block

~~Furthermore, these structured control flow constructs are additionally defined to exclude all outer constructs’ continue constructs and exclude all blocks dominated by all outer constructs’ merge blocks.~~

Loop with early break: no problem!

```
void main() {
  int i = 0; // %1
  do { // %2
    if (i > 50) { // %3
      break;
    } else {
      // no-op // %4
    }
    i++; // %5
  }
  while(i < 100); // %6
  // %7
}
```



Selection construct headed at 3?

Blocks structurally dominated by 3

{3, 4, 5}

Minus blocks structurally dominated by 3’s merge - i.e. by 5

{5}

Yields {3, 4}

Structural dominance allows for simple, intuitive definitions

POPL 2023 paper: details many problems that structural dominance solves

SPIR-V spec updated: definitions now based on structural dominance

SPIR-V validator: now check rules based on structural dominance

Google consult with Imperial team regarding follow-on issues

Bonus: a method for compiler fuzzing

Our Alloy model produces weird and wonderful valid SPIR-V control flow graphs

spirv-to-alloy turns these into skeletal SPIR-V code

But: code is **not runnable**

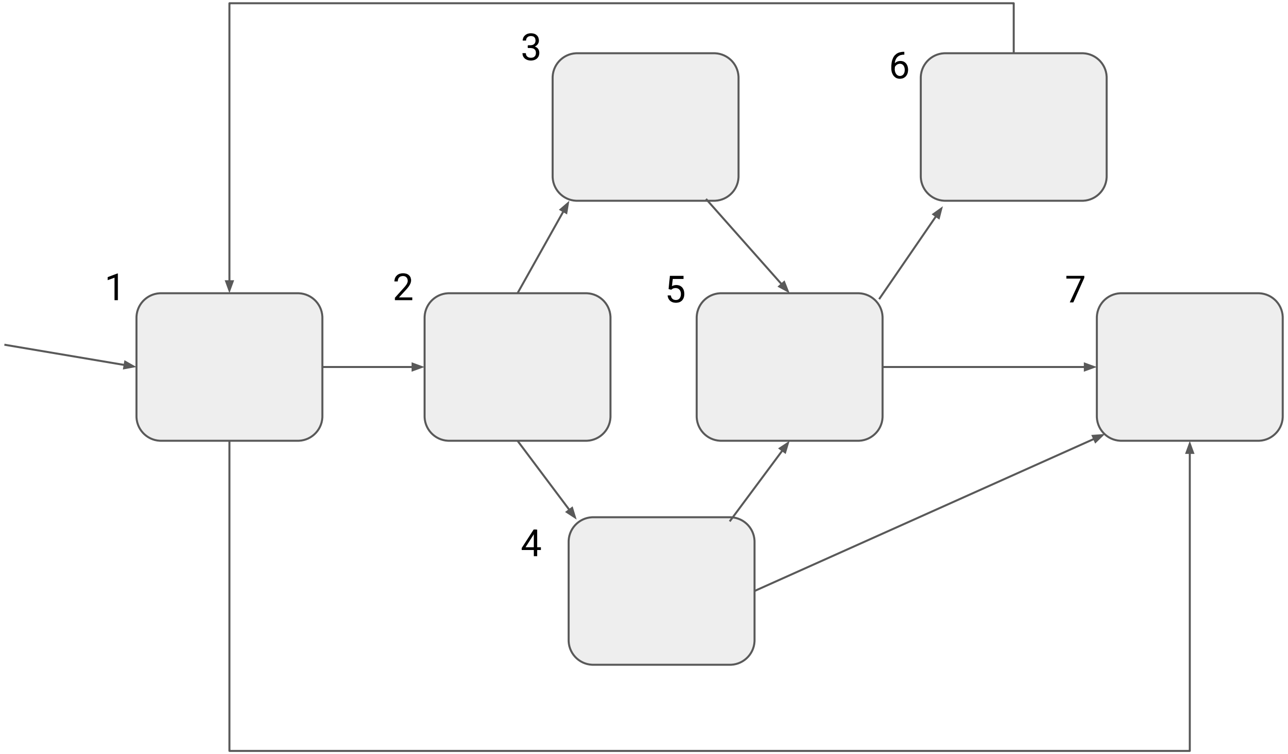
Bonus: a method for compiler fuzzing

Simple idea: **fleshing**

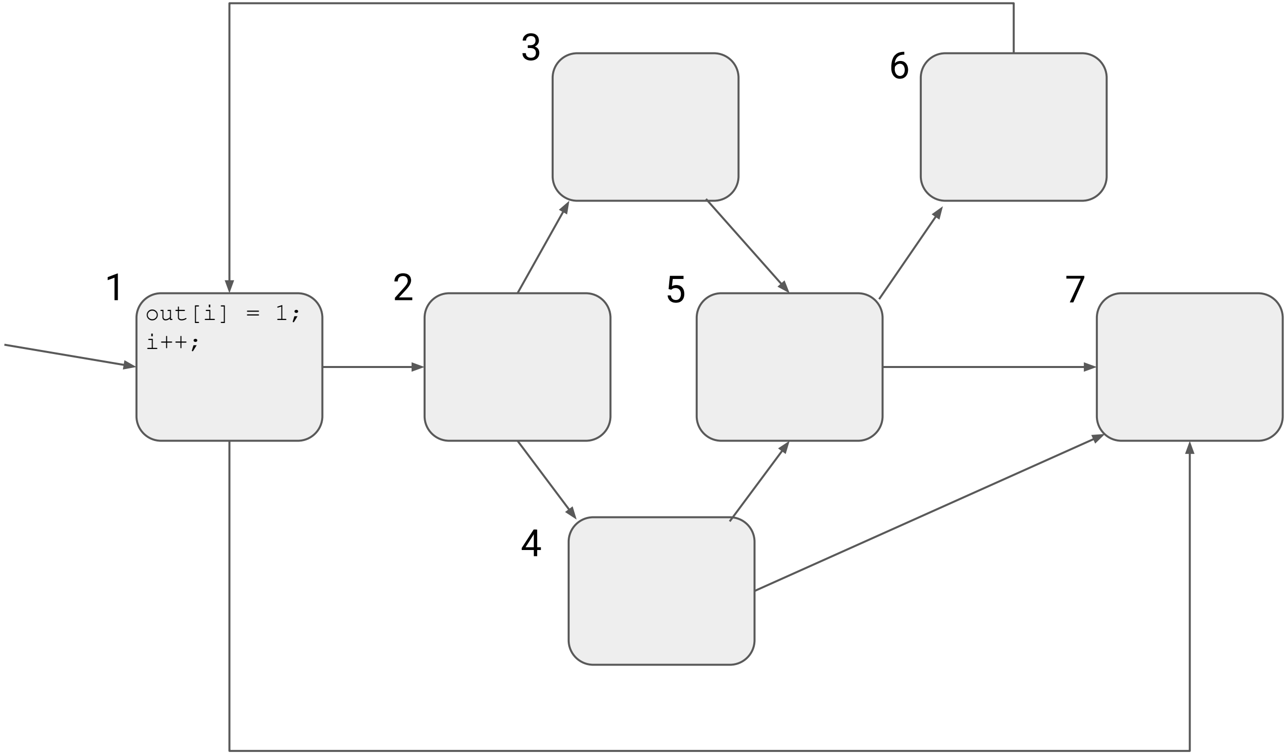
- Instrument CFG so that it (a) follows path dictated by input, and (b) records the path that was followed
- Choose a path through the CFG
- Check that when executed with input that forces this path, the path really is followed

Led to discovery of 20 previously unknown compiler bugs

Fleshing: example

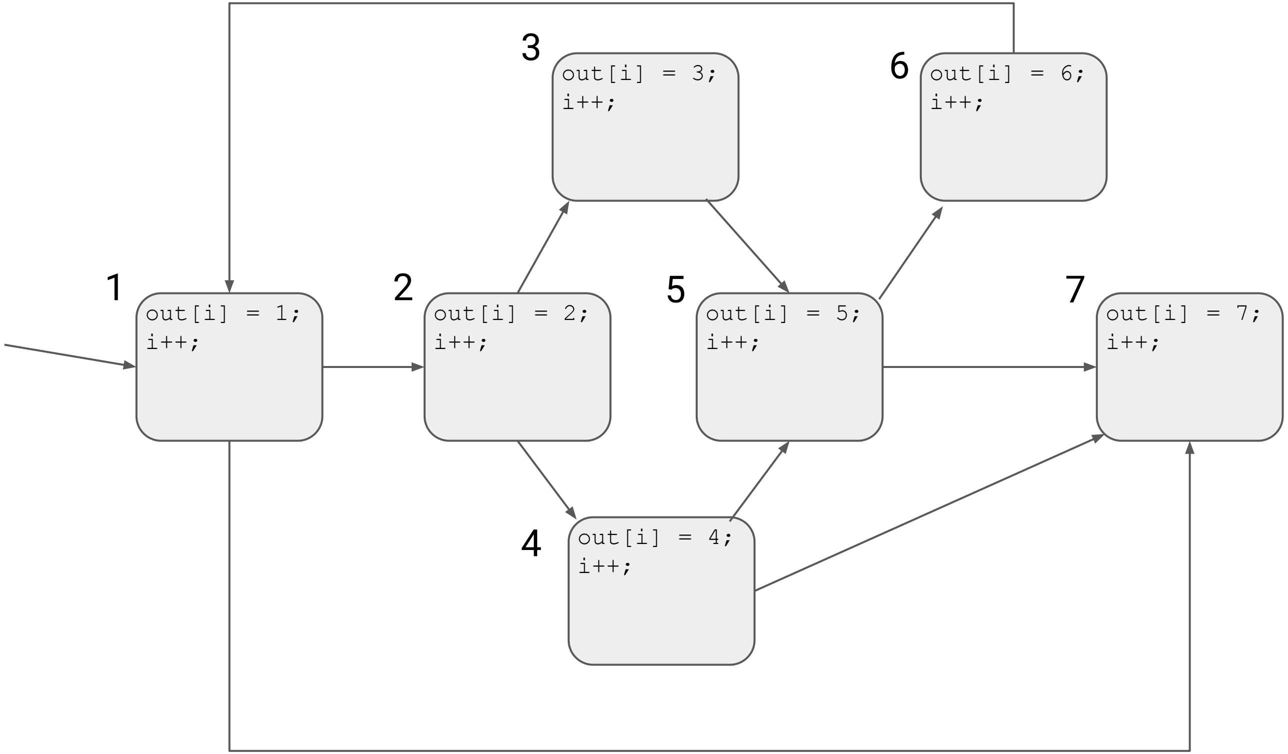


Fleshing: example

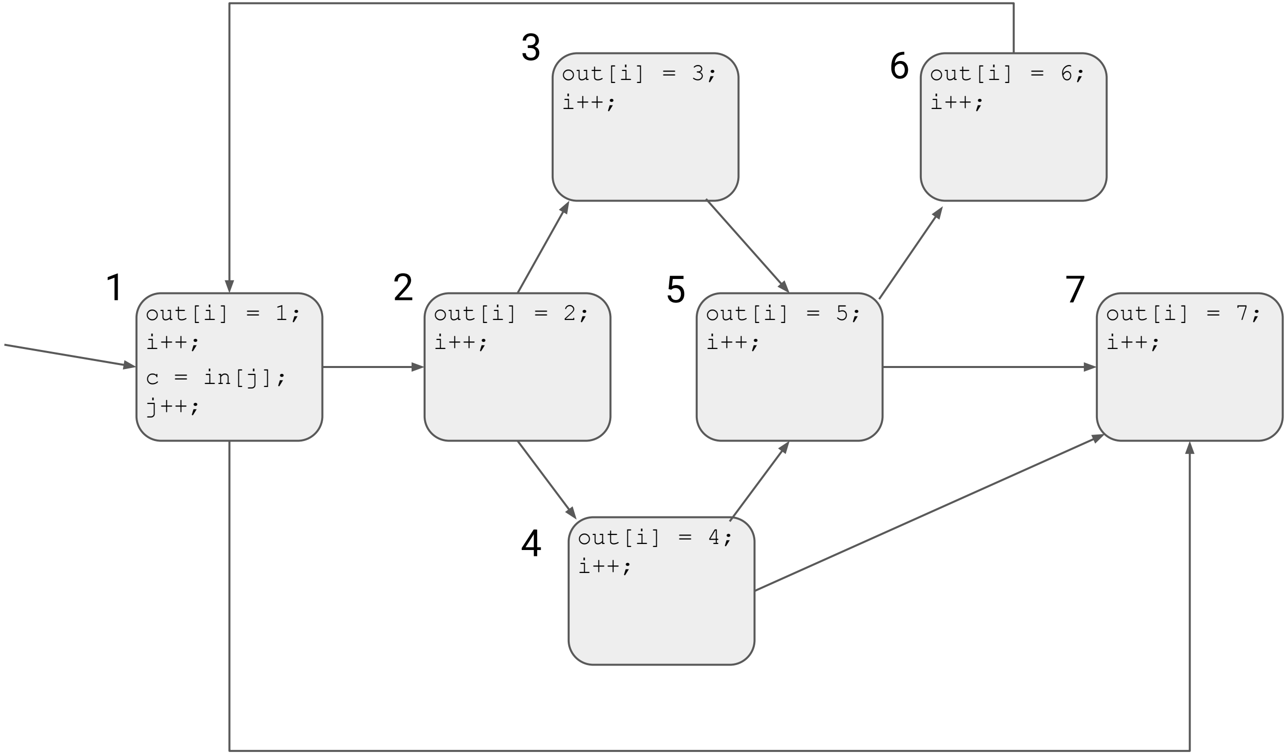


```
out[i] = 1;
i++;
```

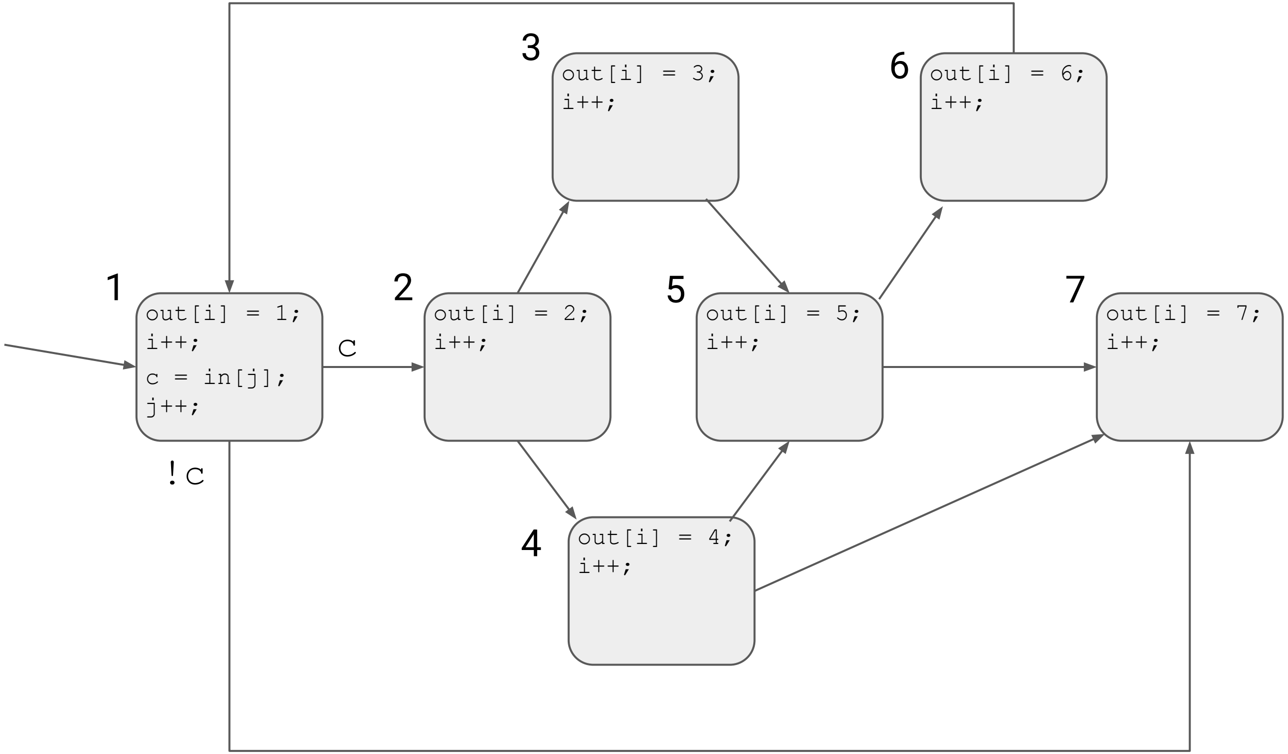
Fleshing: example



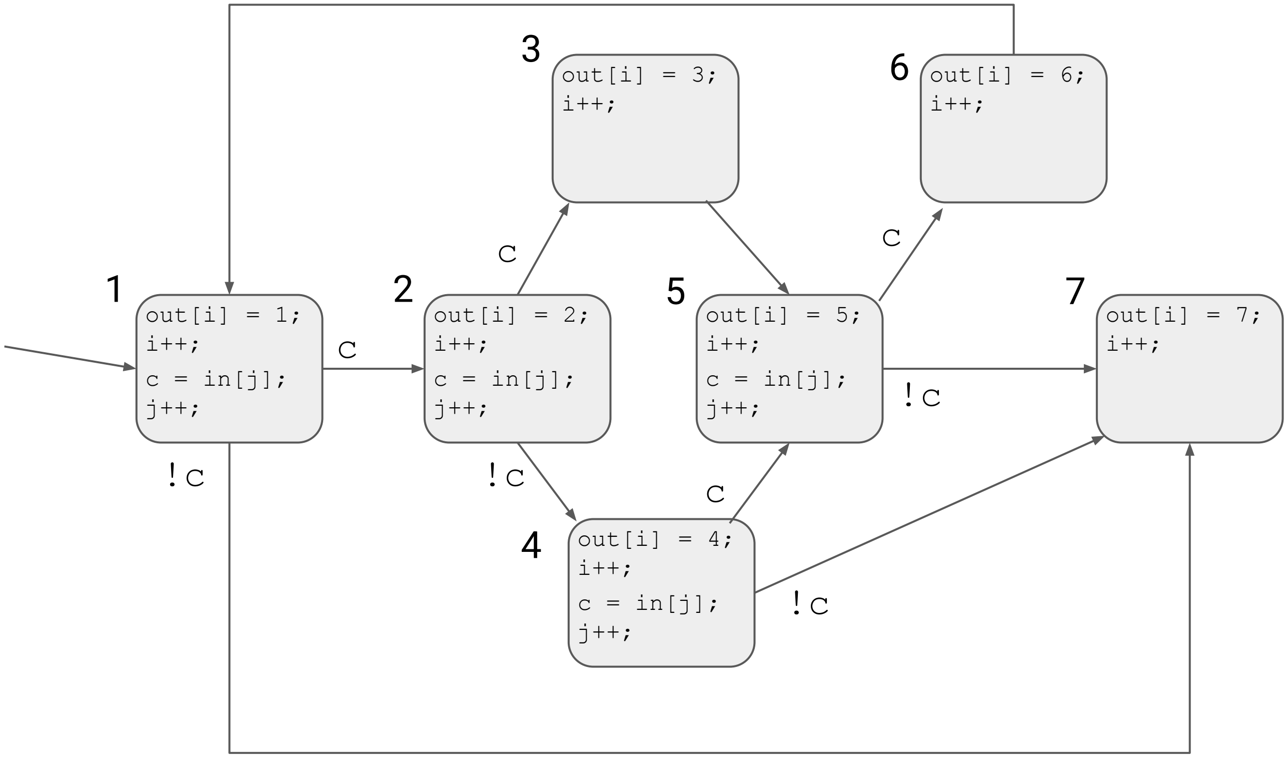
Fleshing: example



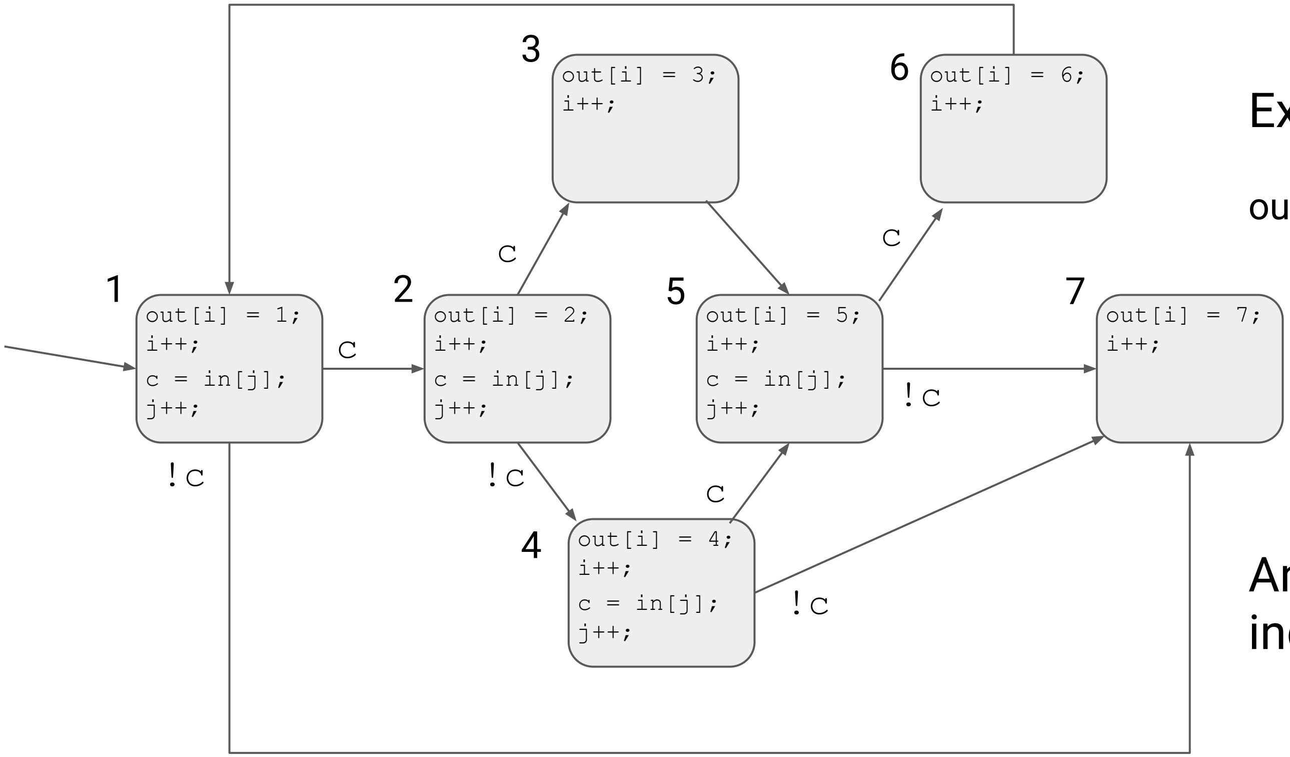
Fleshing: example



Fleshing: example



Fleshing: example



Run program with:

in = [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

Expect:

out = [1, 2, 3, 5, 6,
1, 2, 4, 5, 6,
1, 2, 4, 7]

Any other answer indicates compiler bug

Summary

Formal modelling of SPIR-V allowed fundamental problems to be rectified

Industry have adopted the changes we proposed

Solution - structural dominance - is pleasingly simple, perhaps obvious in hindsight

Our Alloy model can generate challenging CFGs that, after fleshing, can trigger compiler bugs

Future work:

- Does structural dominance have broader relevance? Perhaps not.
- Can fleshing be used to find bugs in compilers for other languages?