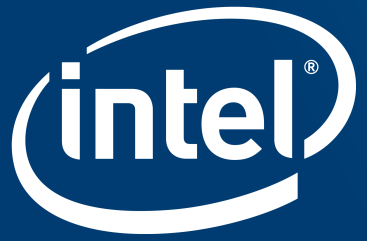
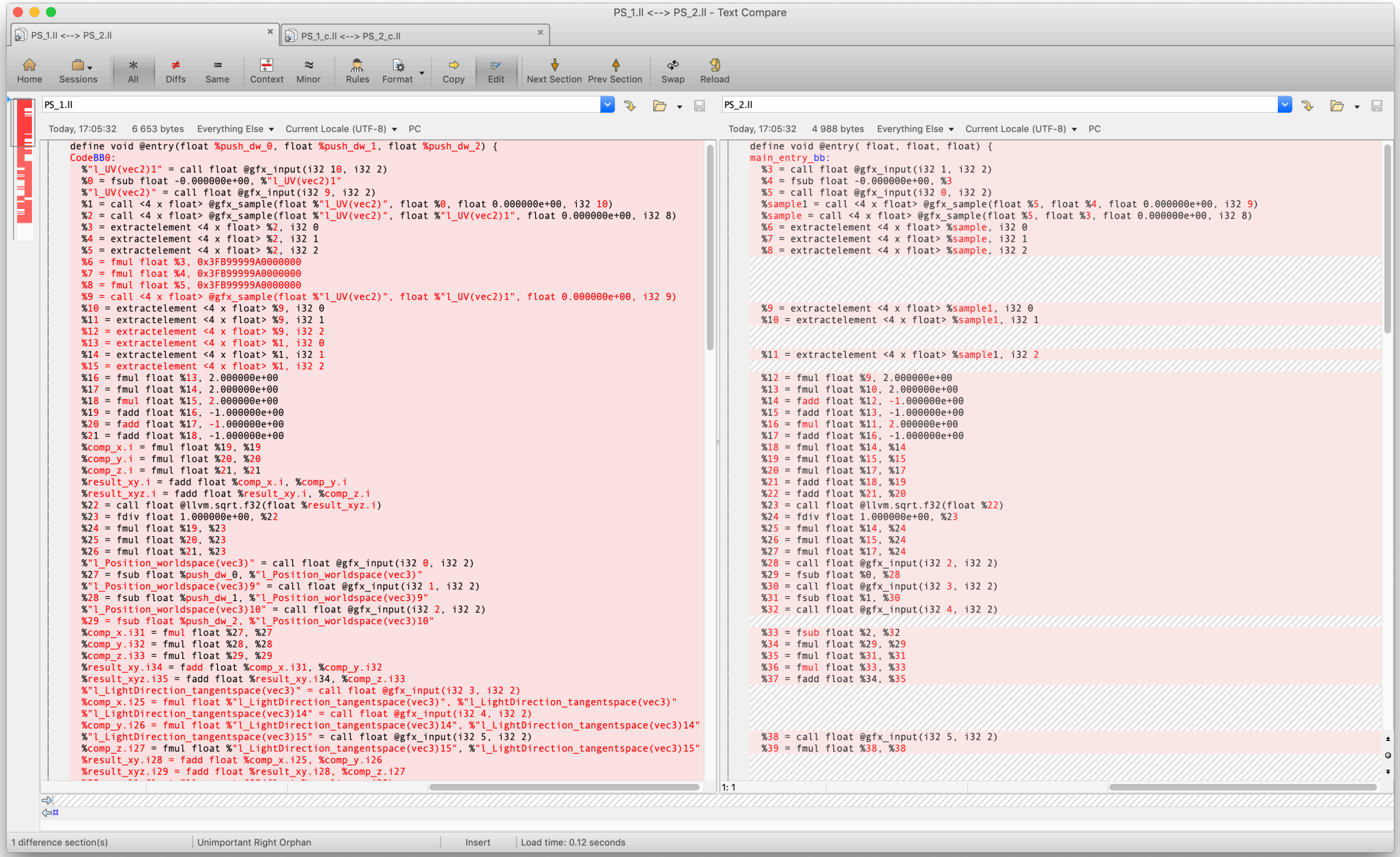


LLVM-CANON: SHOOTING FOR CLEAR DIFFS

Presenting: Michal Paszkowski
Research: Michal Paszkowski, Radoslaw Drabinski
Special thanks to: Julia Koval


PS_1.ll <--> PS_2.ll - Text Compare
PS_1.ll <--> PS_2.ll
PS_1_c.ll <--> PS_2_c.ll
Home
Sessions
All
Diffs
Same
Context
Minor
Rules
Format
Copy
Edit
Next Section
Prev Section
Swap
Reload
PS_1.ll
PS_2.ll
Today, 17:05:32 6 653 bytes Everything Else Current Locale (UTF-8) PC
Today, 17:05:32 4 988 bytes Everything Else Current Locale (UTF-8) PC
define void @entry(float %push_dw_0, float %push_dw_1, float %push_dw_2) {
CodeBB0:
%"l_UV(vec2)1" = call float @gfx_input(i32 10, i32 2)
%0 = fsub float -0.000000e+00, %"l_UV(vec2)1"
%"l_UV(vec2)" = call float @gfx_input(i32 9, i32 2)
%1 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %0, float 0.000000e+00, i32 10)
%2 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %"l_UV(vec2)1", float 0.000000e+00, i32 8)
%3 = extractelement <4 x float> %2, i32 0
%4 = extractelement <4 x float> %2, i32 1
%5 = extractelement <4 x float> %2, i32 2
%6 = fmul float %3, 0x3FB99999A0000000
%7 = fmul float %4, 0x3FB99999A0000000
%8 = fmul float %5, 0x3FB99999A0000000
%9 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %"l_UV(vec2)1", float 0.000000e+00, i32 9)
%10 = extractelement <4 x float> %9, i32 0
%11 = extractelement <4 x float> %9, i32 1
%12 = extractelement <4 x float> %9, i32 2
%13 = extractelement <4 x float> %1, i32 0
%14 = extractelement <4 x float> %1, i32 1
%15 = extractelement <4 x float> %1, i32 2
%16 = fmul float %13, 2.000000e+00
%17 = fmul float %14, 2.000000e+00
%18 = fmul float %15, 2.000000e+00
%19 = fadd float %16, -1.000000e+00
%20 = fadd float %17, -1.000000e+00
%21 = fadd float %18, -1.000000e+00
%comp_x.i = fmul float %19, %19
%comp_y.i = fmul float %20, %20
%comp_z.i = fmul float %21, %21
%result_xy.i = fadd float %comp_x.i, %comp_y.i
%result_xyz.i = fadd float %result_xy.i, %comp_z.i
%22 = call float @llvm.sqrt.f32(float %result_xyz.i)
%23 = fdiv float 1.000000e+00, %22
%24 = fmul float %19, %23
%25 = fmul float %20, %23
%26 = fmul float %21, %23
%"l_Position_worldspace(vec3)" = call float @gfx_input(i32 0, i32 2)
%27 = fsub float %push_dw_0, %"l_Position_worldspace(vec3)"
%"l_Position_worldspace(vec3)9" = call float @gfx_input(i32 1, i32 2)
%28 = fsub float %push_dw_1, %"l_Position_worldspace(vec3)9"
%"l_Position_worldspace(vec3)10" = call float @gfx_input(i32 2, i32 2)
%29 = fsub float %push_dw_2, %"l_Position_worldspace(vec3)10"
%comp_x.i31 = fmul float %27, %27
%comp_y.i32 = fmul float %28, %28
%comp_z.i33 = fmul float %29, %29
%result_xy.i34 = fadd float %comp_x.i31, %comp_y.i32
%result_xyz.i35 = fadd float %result_xy.i34, %comp_z.i33
%"l_LightDirection_tangentspace(vec3)" = call float @gfx_input(i32 3, i32 2)
%comp_x.i25 = fmul float %"l_LightDirection_tangentspace(vec3)", %"l_LightDirection_tangentspace(vec3)"
%"l_LightDirection_tangentspace(vec3)14" = call float @gfx_input(i32 4, i32 2)
%comp_y.i26 = fmul float %"l_LightDirection_tangentspace(vec3)14", %"l_LightDirection_tangentspace(vec3)14"
%"l_LightDirection_tangentspace(vec3)15" = call float @gfx_input(i32 5, i32 2)
%comp_z.i27 = fmul float %"l_LightDirection_tangentspace(vec3)15", %"l_LightDirection_tangentspace(vec3)15"
%result_xy.i28 = fadd float %comp_x.i25, %comp_y.i26
%result_xyz.i29 = fadd float %result_xy.i28, %comp_z.i27
define void @entry(float, float, float) {
main_entry_bb:
%3 = call float @gfx_input(i32 1, i32 2)
%4 = fsub float -0.000000e+00, %3
%5 = call float @gfx_input(i32 0, i32 2)
%sample1 = call <4 x float> @gfx_sample(float %5, float %4, float 0.000000e+00, i32 9)
%sample = call <4 x float> @gfx_sample(float %5, float %3, float 0.000000e+00, i32 8)
%6 = extractelement <4 x float> %sample, i32 0
%7 = extractelement <4 x float> %sample, i32 1
%8 = extractelement <4 x float> %sample, i32 2
%9 = extractelement <4 x float> %sample1, i32 0
%10 = extractelement <4 x float> %sample1, i32 1
%11 = extractelement <4 x float> %sample1, i32 2
%12 = fmul float %9, 2.000000e+00
%13 = fmul float %10, 2.000000e+00
%14 = fadd float %12, -1.000000e+00
%15 = fadd float %13, -1.000000e+00
%16 = fmul float %11, 2.000000e+00
%17 = fadd float %16, -1.000000e+00
%18 = fmul float %14, %14
%19 = fmul float %15, %15
%20 = fmul float %17, %17
%21 = fadd float %18, %19
%22 = fadd float %21, %20
%23 = call float @llvm.sqrt.f32(float %22)
%24 = fdiv float 1.000000e+00, %23
%25 = fmul float %14, %24
%26 = fmul float %15, %24
%27 = fmul float %17, %24
%28 = call float @gfx_input(i32 2, i32 2)
%29 = fsub float %0, %28
%30 = call float @gfx_input(i32 3, i32 2)
%31 = fsub float %1, %30
%32 = call float @gfx_input(i32 4, i32 2)
%33 = fsub float %2, %32
%34 = fmul float %29, %29
%35 = fmul float %31, %31
%36 = fmul float %33, %33
%37 = fadd float %34, %35
%38 = call float @gfx_input(i32 5, i32 2)
%39 = fmul float %38, %38
1: 1
1 difference section(s)
Unimportant Right Orphan
Insert
Load time: 0.12 seconds

It is all about the semantic differences!

Desired

- Semantics

Distracting

- Instruction order
- Value names
- Basic Block names
- Redundant code

How could we automate that?

We needed a simple tool that would makes comparing semantic differences between two modules easier.

The tool should…

- ”Reduce” non-semantic differences
- Process modules independently
- Leverage existing diff tools

How could we automate that?

Therefore, the tool should transform a module into a canonical form.

How will that “canonical form” help us?

- Two semantically identical canonicalized modules should show no differences when diffed.

And more importantly…

- When the modules are not identical the semantic differences should stand out.

How do we arrive at this canonical form?

Let’s start with instruction ordering…

Assumptions for comparing an identical module after two different transformations:

- Side-Effects should be roughly the same
- Control-Flow Graphs should be similar

Instruction reordering

- *def-use* distance reduction

```
%1 = ...

%2 = ...

%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1

%C = fsub float %T1, %B

%X = fadd float %C, %A

store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

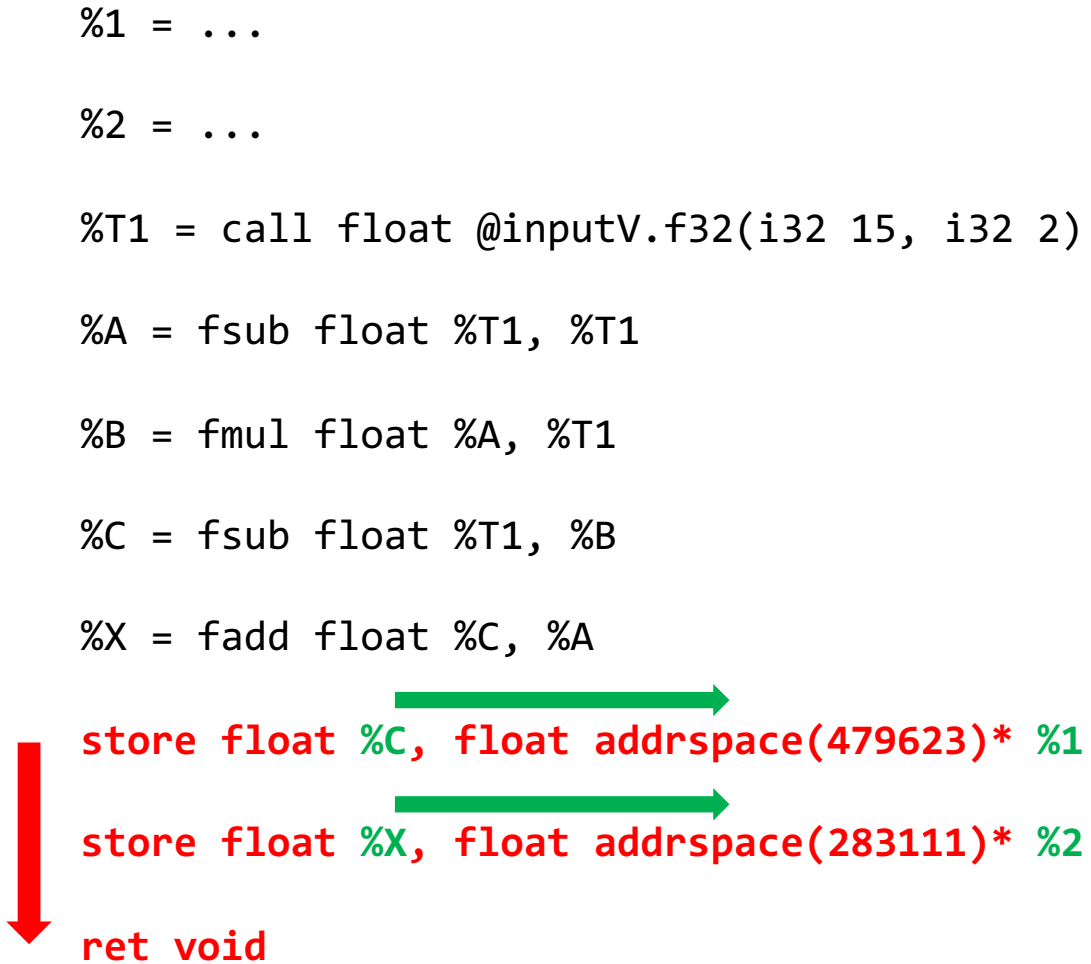
Instruction reordering

- *def-use* distance reduction
  1. Collect instructions with side-effects and ”ret” instructions

```
%1 = ...

%2 = ...

%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1

%C = fsub float %T1, %B

%X = fadd float %C, %A

store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Instruction reordering

- *def-use* distance reduction
  1. Collect instructions with side-effects and ”ret” instructions
  2. Walk the instructions with side-effects (top-down) and on each instruction their operands (left-right)



Instruction reordering

- *def-use* distance reduction
  1. Collect instructions with side-effects and ”ret” instructions
  2. Walk the instructions with side-effects (top-down) and on each instruction their operands (left-right)
  3. For each operand, bring their definition as close as possible to the using instruction

```
%1 = ...

%2 = ...

%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1

%C = fsub float %T1, %B

%X = fadd float %C, %A

store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1


%C = fsub float %T1, %B
%X = fadd float %C, %A




store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1


%C = fsub float %T1, %B
%X = fadd float %C, %A




store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 1st side-effecting instruction.
Don't move it, otherwise semantics may not be preserved!

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1


%C = fsub float %T1, %B
%X = fadd float %C, %A




store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Take the 1st operand of the side-effecting instruction.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1


%C = fsub float %T1, %B
%X = fadd float %C, %A
```

```
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Move it closer to the user.
Def-Use sequence may be temporarily broken.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1



%X = fadd float %C, %A



%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 1st operand of the moved instruction.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%T1 = call float @inputV.f32(i32 15, i32 2)

%A = fsub float %T1, %T1

%B = fmul float %A, %T1




%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)



%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Move it closer to the user.
Def-Use sequence may be temporarily broken.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

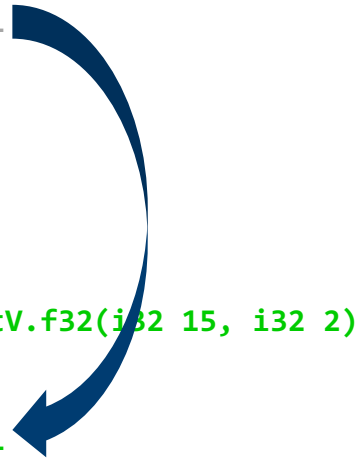
```
%A = fsub float %T1, %T1

%B = fmul float %A, %T1



%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)


%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 2nd operand of the previously moved instruction.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%A = fsub float %T1, %T1

%B = fmul float %A, %T1



%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)

%B = fmul float %A, %T1
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```



Move it closer to the user.
Def-Use sequence is being repaired.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

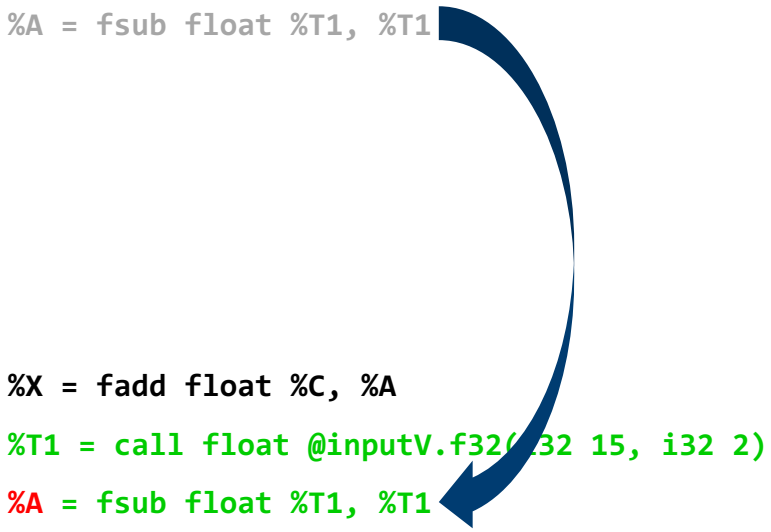
```
%A = fsub float %T1, %T1
```

```
%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)

%B = fmul float %A, %T1
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 1st operand of the moved instruction.

● **Canonicalized instructions**

Step-by-step reorder walkthrough



Move it closer to the user.
Def-Use sequence is being repaired.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)
%A = fsub float %T1, %T1
%B = fmul float %A, %T1
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 1st operand of the moved instruction.
Don't move it! %T1 has been already moved!

Select the 2nd operand of the moved instruction.
Don't move it! %T1 has been already moved!

● **Canonicalized instructions**

Step-by-step reorder walkthrough

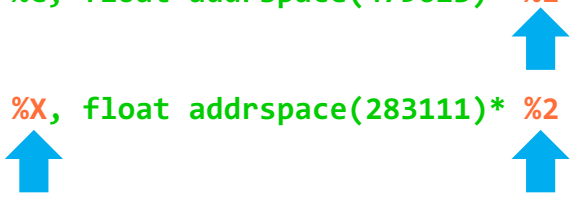
```
%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)
%A = fsub float %T1, %T1
%B = fmul float %A, %T1
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1

store float %X, float addrspace(283111)* %2

ret void
```

Select the 2nd operand of the previously moved instruction.
Don't move it, %T1 has been moved before.

● **Canonicalized instructions**

Step-by-step reorder walkthrough

```
%X = fadd float %C, %A
%T1 = call float @inputV.f32(i32 15, i32 2)
%A = fsub float %T1, %T1
%B = fmul float %A, %T1
%C = fsub float %T1, %B
store float %C, float addrspace(479623)* %1
store float %X, float addrspace(283111)* %2
ret void
```



We repeat the process for all operands in all side-effecting instructions.

● **Canonicalized instructions**

How do we arrive at this canonical form?

Desired

- Semantics

Distracting

- Instruction order ✓
- Value names
- Basic Block names
- Redundant code

Naming instructions: Linear

- Numbers all instructions top-down after reordering vn
- We were hoping that maybe the reordering mechanism could be used as a 'seed' for instruction naming

```
%T1 = @inputV.f32(i32 15, i32 2)
%A = fsub float %T1, %T1
%B = fmul float %A, %T1
%C = fsub float %T1, %T1
%X = fadd float %C, %A
store float %C, float addrspace(479623)* %1
store float %X, float addrspace(283111)* %2
ret void
```

```
%v0 = @inputV.f32(i32 15, i32 2)
%v1 = fsub float %v0, %v0
%v2 = fmul float %v1, %v0
%v3 = fsub float %v0, %v0
%v4 = fadd float %v3, %v1
store float %v3, float addrspace(479623)* %a3
store float %v4, float addrspace(283111)* %a4
ret void
```

Naming instructions: Linear

FAILURE

- Numbers all instructions top-down after reordering vn
- We were hoping that maybe the reordering mechanism could be us
  'seed' for instruction naming

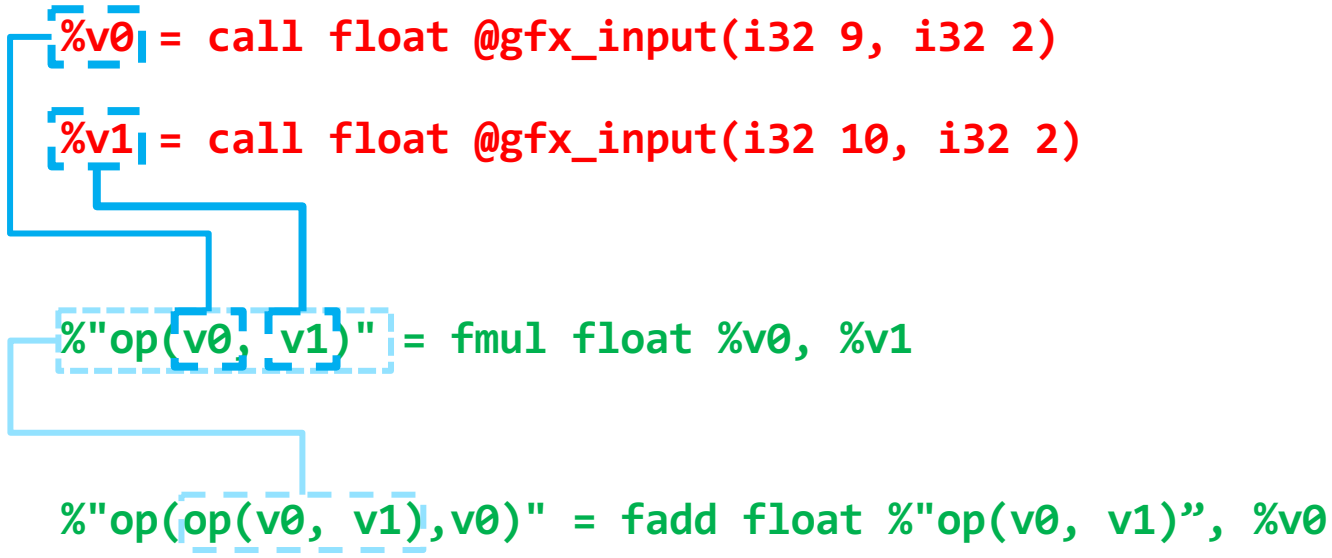
```
%T1 = @inputV.f32(i32 15, i32 2)
%A = fsub float %T1, %T1
%B = fmul float %A, %T1
%C = fsub float %T1, %T1
%X = fadd float %C, %A
store float %C, float addrspace(479623)* %1
store float %X, float addrspace(283111)* %2
ret void
```

```
%v0 = @inputV.f32(i32 15, i32 2)
%v1 = fsub float %v0, %v0
%v2 = fmul float %v1, %v0
%v3 = fsub float %v0, %v0
%v4 = fadd float %v3, %v1
store float %v3, float addrspace(479623)* %a3
store float %v4, float addrspace(283111)* %a4
ret void
```

Naming instructions: “Graph naming”

Two types of instructions:

1. Initial instructions
   - Instructions with only immediate operands
   - Numbered according to positions of outputs using that instruction after sorting
2. Regular instructions
   - “Graph naming”: Differences in defs are reflected in uses

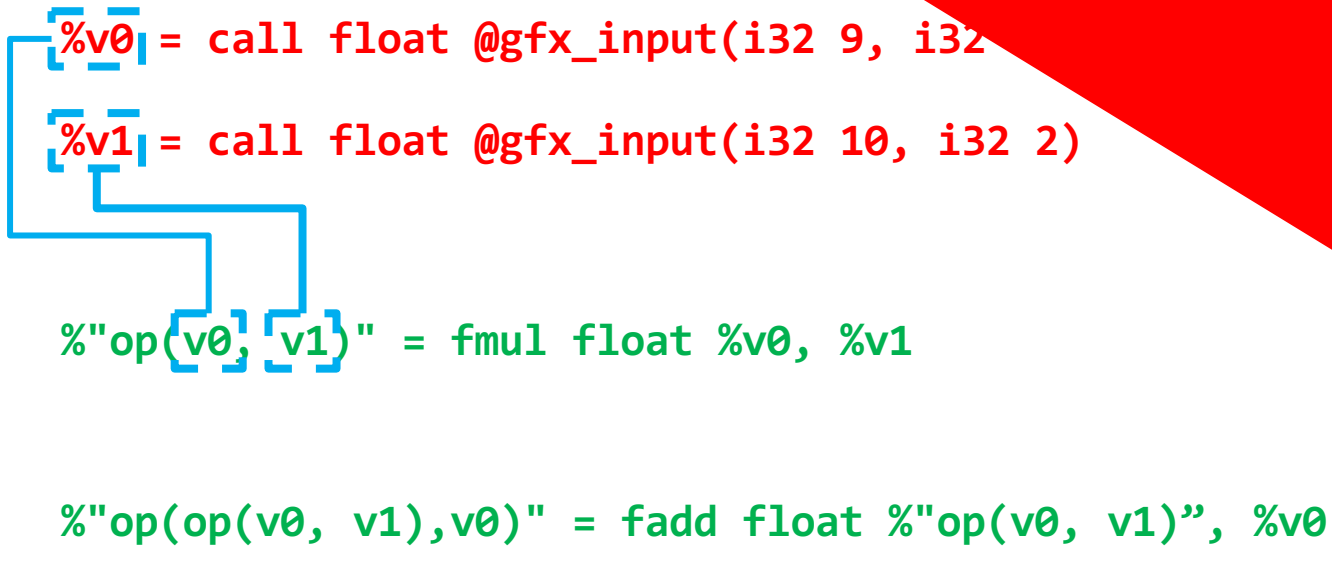


Naming instructions: “Graph nam

ALMOST THERE

Two types of instructions:

1. Initial instructions
   - Instructions with only immediate operands
   - Numbered according to positions of outputs using that instruction after sorting
2. Regular instructions
   - “Graph naming”: Differences in defs are reflected in uses



Naming instructions: “Graph nam

ALMOST THERE

Two types of instructions:

1. Initial instructions
   - Instructions with only immediate operands
   - Numbered according to positions of outputs using that instruction after sorting

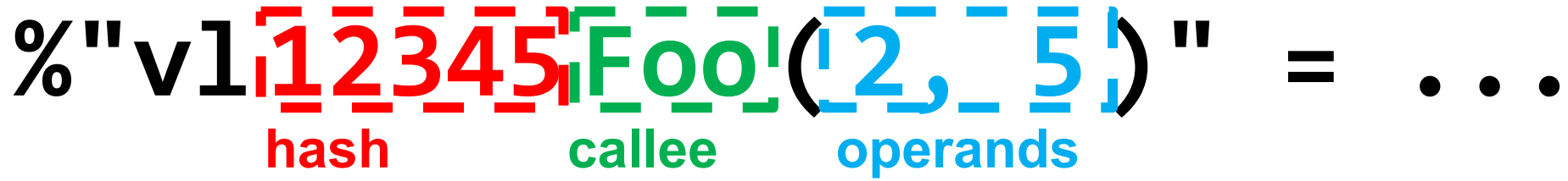
Instructions with different opcodes but same users got the same names.

2. Regular instructions
   - “Graph naming”: Differences in defs are reflected in uses

Extremely long names!
Differences in defs should be reflected only in outputs

Naming instructions: Current version

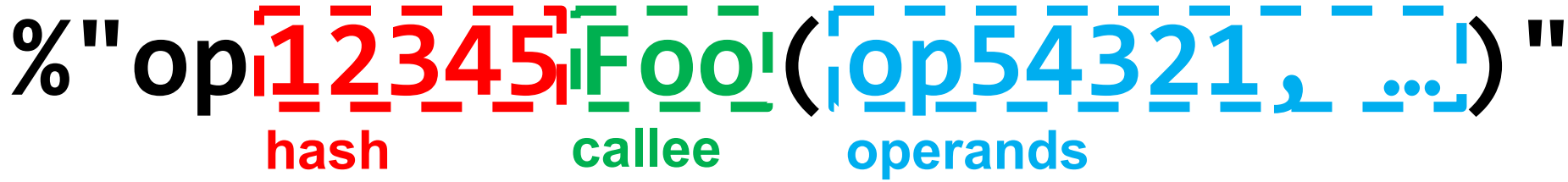
1. Initial instructions (those with only immediate operands)



- Hash calculated considering instruction's opcode and the "output footprint"
- Called function name only included in case of a CallInst
- Immediate operands list (sorted in case of commutative instructions)

Naming instructions: Current version

2. Regular instructions



- Hash calculated considering instruction's and its operands' opcodes
- Called function name only included in case of a CallInst
- Short operand names

Naming instructions: Current version

3. Output instructions (instructions with side-effects and their relative operands)

```
%"op12345 Foo ( op54321(…), … )"
```

- Same as regular instructions, but…
- recursively generated long operand list is kept, so…
- by just looking at an output we see what impacts its semantics in the diff.

Naming instructions: Current version

```
%"vl20713gfx_input(0, 2)" =
```

Initial instructions

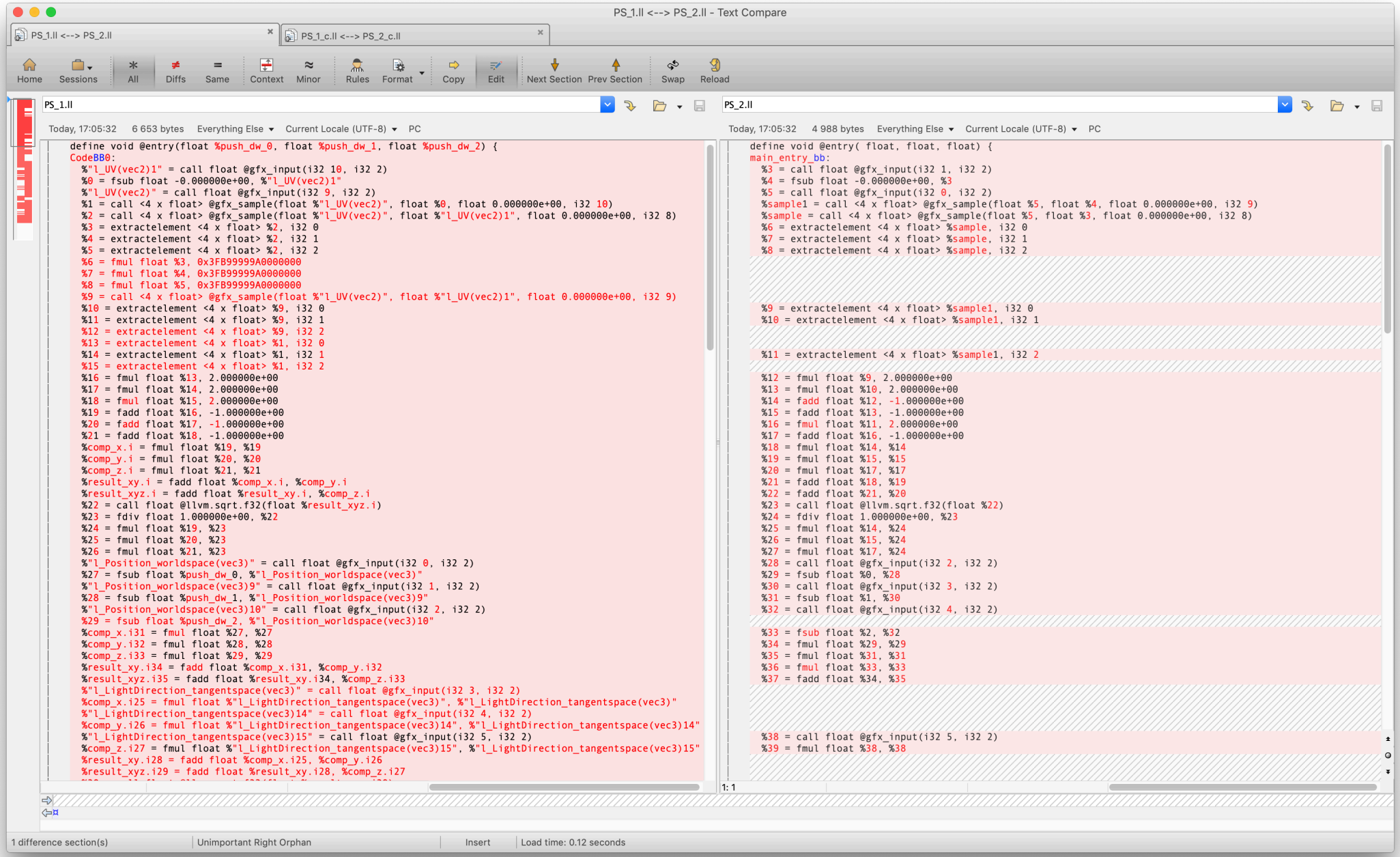
```
%"vl115160gfx_input(1, 2)" =
```

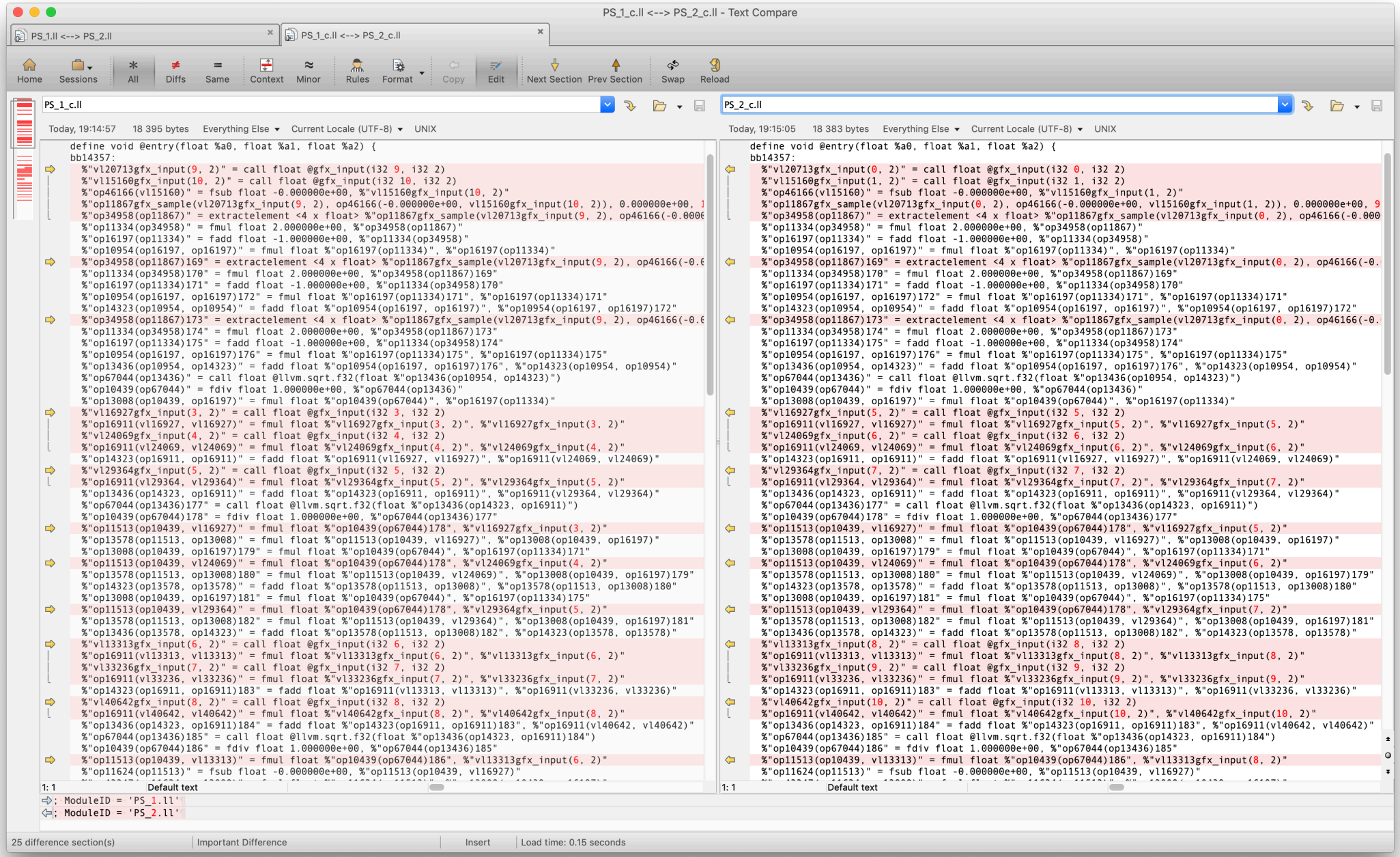
```
%"op46166(vl115160)" =
```

Regular instruction

```
%"op11867(vl20713gfx_input(0, 2), op46166(vl115160…)…) =
```

Output instruction


PS_1.ll <--> PS_2.ll - Text Compare
PS_1.ll <--> PS_2.ll
PS_1_c.ll <--> PS_2_c.ll
Home
Sessions
All
Diffs
Same
Context
Minor
Rules
Format
Copy
Edit
Next Section
Prev Section
Swap
Reload
PS_1.ll
PS_2.ll
Today, 17:05:32 6 653 bytes Everything Else Current Locale (UTF-8) PC
Today, 17:05:32 4 988 bytes Everything Else Current Locale (UTF-8) PC
define void @entry(float %push_dw_0, float %push_dw_1, float %push_dw_2) {
CodeBB0:
%"l_UV(vec2)1" = call float @gfx_input(i32 10, i32 2)
%0 = fsub float -0.000000e+00, %"l_UV(vec2)1"
%"l_UV(vec2)" = call float @gfx_input(i32 9, i32 2)
%1 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %0, float 0.000000e+00, i32 10)
%2 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %"l_UV(vec2)1", float 0.000000e+00, i32 8)
%3 = extractelement <4 x float> %2, i32 0
%4 = extractelement <4 x float> %2, i32 1
%5 = extractelement <4 x float> %2, i32 2
%6 = fmul float %3, 0x3FB99999A0000000
%7 = fmul float %4, 0x3FB99999A0000000
%8 = fmul float %5, 0x3FB99999A0000000
%9 = call <4 x float> @gfx_sample(float %"l_UV(vec2)", float %"l_UV(vec2)1", float 0.000000e+00, i32 9)
%10 = extractelement <4 x float> %9, i32 0
%11 = extractelement <4 x float> %9, i32 1
%12 = extractelement <4 x float> %9, i32 2
%13 = extractelement <4 x float> %1, i32 0
%14 = extractelement <4 x float> %1, i32 1
%15 = extractelement <4 x float> %1, i32 2
%16 = fmul float %13, 2.000000e+00
%17 = fmul float %14, 2.000000e+00
%18 = fmul float %15, 2.000000e+00
%19 = fadd float %16, -1.000000e+00
%20 = fadd float %17, -1.000000e+00
%21 = fadd float %18, -1.000000e+00
%comp_x.i = fmul float %19, %19
%comp_y.i = fmul float %20, %20
%comp_z.i = fmul float %21, %21
%result_xy.i = fadd float %comp_x.i, %comp_y.i
%result_xyz.i = fadd float %result_xy.i, %comp_z.i
%22 = call float @llvm.sqrt.f32(float %result_xyz.i)
%23 = fdiv float 1.000000e+00, %22
%24 = fmul float %19, %23
%25 = fmul float %20, %23
%26 = fmul float %21, %23
%"l_Position_worldspace(vec3)" = call float @gfx_input(i32 0, i32 2)
%27 = fsub float %push_dw_0, %"l_Position_worldspace(vec3)"
%"l_Position_worldspace(vec3)9" = call float @gfx_input(i32 1, i32 2)
%28 = fsub float %push_dw_1, %"l_Position_worldspace(vec3)9"
%"l_Position_worldspace(vec3)10" = call float @gfx_input(i32 2, i32 2)
%29 = fsub float %push_dw_2, %"l_Position_worldspace(vec3)10"
%comp_x.i31 = fmul float %27, %27
%comp_y.i32 = fmul float %28, %28
%comp_z.i33 = fmul float %29, %29
%result_xy.i34 = fadd float %comp_x.i31, %comp_y.i32
%result_xyz.i35 = fadd float %result_xy.i34, %comp_z.i33
%"l_LightDirection_tangentspace(vec3)" = call float @gfx_input(i32 3, i32 2)
%comp_x.i25 = fmul float %"l_LightDirection_tangentspace(vec3)", %"l_LightDirection_tangentspace(vec3)"
%"l_LightDirection_tangentspace(vec3)14" = call float @gfx_input(i32 4, i32 2)
%comp_y.i26 = fmul float %"l_LightDirection_tangentspace(vec3)14", %"l_LightDirection_tangentspace(vec3)14"
%"l_LightDirection_tangentspace(vec3)15" = call float @gfx_input(i32 5, i32 2)
%comp_z.i27 = fmul float %"l_LightDirection_tangentspace(vec3)15", %"l_LightDirection_tangentspace(vec3)15"
%result_xy.i28 = fadd float %comp_x.i25, %comp_y.i26
%result_xyz.i29 = fadd float %result_xy.i28, %comp_z.i27
define void @entry(float, float, float) {
main_entry_bb:
%3 = call float @gfx_input(i32 1, i32 2)
%4 = fsub float -0.000000e+00, %3
%5 = call float @gfx_input(i32 0, i32 2)
%sample1 = call <4 x float> @gfx_sample(float %5, float %4, float 0.000000e+00, i32 9)
%sample = call <4 x float> @gfx_sample(float %5, float %3, float 0.000000e+00, i32 8)
%6 = extractelement <4 x float> %sample, i32 0
%7 = extractelement <4 x float> %sample, i32 1
%8 = extractelement <4 x float> %sample, i32 2
%9 = extractelement <4 x float> %sample1, i32 0
%10 = extractelement <4 x float> %sample1, i32 1
%11 = extractelement <4 x float> %sample1, i32 2
%12 = fmul float %9, 2.000000e+00
%13 = fmul float %10, 2.000000e+00
%14 = fadd float %12, -1.000000e+00
%15 = fadd float %13, -1.000000e+00
%16 = fmul float %11, 2.000000e+00
%17 = fadd float %16, -1.000000e+00
%18 = fmul float %14, %14
%19 = fmul float %15, %15
%20 = fmul float %17, %17
%21 = fadd float %18, %19
%22 = fadd float %21, %20
%23 = call float @llvm.sqrt.f32(float %22)
%24 = fdiv float 1.000000e+00, %23
%25 = fmul float %14, %24
%26 = fmul float %15, %24
%27 = fmul float %17, %24
%28 = call float @gfx_input(i32 2, i32 2)
%29 = fsub float %0, %28
%30 = call float @gfx_input(i32 3, i32 2)
%31 = fsub float %1, %30
%32 = call float @gfx_input(i32 4, i32 2)
%33 = fsub float %2, %32
%34 = fmul float %29, %29
%35 = fmul float %31, %31
%36 = fmul float %33, %33
%37 = fadd float %34, %35
%38 = call float @gfx_input(i32 5, i32 2)
%39 = fmul float %38, %38
1: 1
1 difference section(s)
Unimportant Right Orphan
Insert
Load time: 0.12 seconds


PS_1_c.ll <--> PS_2_c.ll - Text Compare
PS_1.ll <--> PS_2.ll
PS_1_c.ll <--> PS_2_c.ll
Home
Sessions
All
Diffs
Same
Context
Minor
Rules
Format
Copy
Edit
Next Section
Prev Section
Swap
Reload
PS_1_c.ll
PS_2_c.ll
Today, 19:14:57
18 395 bytes
Everything Else
Current Locale (UTF-8)
UNIX
Today, 19:15:05
18 383 bytes
Everything Else
Current Locale (UTF-8)
UNIX
define void @entry(float %a0, float %a1, float %a2) {
bb14357:
%"vl20713gfx_input(9, 2)" = call float @gfx_input(i32 9, i32 2)
%"vl15160gfx_input(10, 2)" = call float @gfx_input(i32 10, i32 2)
%"op46166(vl15160)" = fsub float -0.000000e+00, %"vl15160gfx_input(10, 2)"
%"op11867gfx_sample(vl20713gfx_input(9, 2), op46166(-0.000000e+00, vl15160gfx_input(10, 2)), 0.000000e+00, 1
%"op34958(op11867)" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(9, 2), op46166(-0.0006
%"op11334(op34958)" = fmul float 2.000000e+00, %"op34958(op11867)"
%"op16197(op11334)" = fadd float -1.000000e+00, %"op11334(op34958)"
%"op10954(op16197, op16197)" = fmul float %"op16197(op11334)", %"op16197(op11334)"
%"op34958(op11867)169" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(9, 2), op46166(-0.6
%"op11334(op34958)170" = fmul float 2.000000e+00, %"op34958(op11867)169"
%"op16197(op11334)171" = fadd float -1.000000e+00, %"op11334(op34958)170"
%"op10954(op16197, op16197)172" = fmul float %"op16197(op11334)171", %"op16197(op11334)171"
%"op14323(op10954, op10954)" = fadd float %"op10954(op16197, op16197)", %"op10954(op16197, op16197)172"
%"op34958(op11867)173" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(9, 2), op46166(-0.6
%"op11334(op34958)174" = fmul float 2.000000e+00, %"op34958(op11867)173"
%"op16197(op11334)175" = fadd float -1.000000e+00, %"op11334(op34958)174"
%"op10954(op16197, op16197)176" = fmul float %"op16197(op11334)175", %"op16197(op11334)175"
%"op13436(op10954, op14323)" = fadd float %"op10954(op16197, op16197)176", %"op14323(op10954, op10954)"
%"op67044(op13436)" = call float @llvm.sqrt.f32(float %"op13436(op10954, op14323)")
%"op10439(op67044)" = fdiv float 1.000000e+00, %"op67044(op13436)"
%"op13008(op10439, op16197)" = fmul float %"op10439(op67044)", %"op16197(op11334)"
%"vl16927gfx_input(3, 2)" = call float @gfx_input(i32 3, i32 2)
%"op16911(vl16927, vl16927)" = fmul float %"vl16927gfx_input(3, 2)", %"vl16927gfx_input(3, 2)"
%"vl24069gfx_input(4, 2)" = call float @gfx_input(i32 4, i32 2)
%"op16911(vl24069, vl24069)" = fmul float %"vl24069gfx_input(4, 2)", %"vl24069gfx_input(4, 2)"
%"op14323(op16911, op16911)" = fadd float %"op16911(vl16927, vl16927)", %"op16911(vl24069, vl24069)"
%"vl29364gfx_input(5, 2)" = call float @gfx_input(i32 5, i32 2)
%"op16911(vl29364, vl29364)" = fmul float %"vl29364gfx_input(5, 2)", %"vl29364gfx_input(5, 2)"
%"op13436(op14323, op16911)" = fadd float %"op14323(op16911, op16911)", %"op16911(vl29364, vl29364)"
%"op67044(op13436)177" = call float @llvm.sqrt.f32(float %"op13436(op14323, op16911)")
%"op10439(op67044)178" = fdiv float 1.000000e+00, %"op67044(op13436)177"
%"op11513(op10439, vl16927)" = fmul float %"op10439(op67044)178", %"vl16927gfx_input(3, 2)"
%"op13578(op11513, op13008)" = fmul float %"op11513(op10439, vl16927)", %"op13008(op10439, op16197)"
%"op13008(op10439, op16197)179" = fmul float %"op10439(op67044)", %"op16197(op11334)171"
%"op11513(op10439, vl24069)" = fmul float %"op10439(op67044)178", %"vl24069gfx_input(4, 2)"
%"op13578(op11513, op13008)180" = fmul float %"op11513(op10439, vl24069)", %"op13008(op10439, op16197)179"
%"op14323(op13578, op13578)" = fadd float %"op13578(op11513, op13008)", %"op13578(op11513, op13008)180"
%"op13008(op10439, op16197)181" = fmul float %"op10439(op67044)", %"op16197(op11334)175"
%"op11513(op10439, vl29364)" = fmul float %"op10439(op67044)178", %"vl29364gfx_input(5, 2)"
%"op13578(op11513, op13008)182" = fmul float %"op11513(op10439, vl29364)", %"op13008(op10439, op16197)181"
%"op13436(op13578, op14323)" = fadd float %"op13578(op11513, op13008)182", %"op14323(op13578, op13578)"
%"vl13313gfx_input(6, 2)" = call float @gfx_input(i32 6, i32 2)
%"op16911(vl13313, vl13313)" = fmul float %"vl13313gfx_input(6, 2)", %"vl13313gfx_input(6, 2)"
%"vl33236gfx_input(7, 2)" = call float @gfx_input(i32 7, i32 2)
%"op16911(vl33236, vl33236)" = fmul float %"vl33236gfx_input(7, 2)", %"vl33236gfx_input(7, 2)"
%"op14323(op16911, op16911)183" = fadd float %"op16911(vl13313, vl13313)", %"op16911(vl33236, vl33236)"
%"vl40642gfx_input(8, 2)" = call float @gfx_input(i32 8, i32 2)
%"op16911(vl40642, vl40642)" = fmul float %"vl40642gfx_input(8, 2)", %"vl40642gfx_input(8, 2)"
%"op13436(op14323, op16911)184" = fadd float %"op14323(op16911, op16911)183", %"op16911(vl40642, vl40642)"
%"op67044(op13436)185" = call float @llvm.sqrt.f32(float %"op13436(op14323, op16911)184")
%"op10439(op67044)186" = fdiv float 1.000000e+00, %"op67044(op13436)185"
%"op11513(op10439, vl13313)" = fmul float %"op10439(op67044)186", %"vl13313gfx_input(6, 2)"
%"op11624(op11513)" = fsub float -0.000000e+00, %"op11513(op10439, vl16927)"
define void @entry(float %a0, float %a1, float %a2) {
bb14357:
%"vl20713gfx_input(0, 2)" = call float @gfx_input(i32 0, i32 2)
%"vl15160gfx_input(1, 2)" = call float @gfx_input(i32 1, i32 2)
%"op46166(vl15160)" = fsub float -0.000000e+00, %"vl15160gfx_input(1, 2)"
%"op11867gfx_sample(vl20713gfx_input(0, 2), op46166(-0.000000e+00, vl15160gfx_input(1, 2)), 0.000000e+00, 9
%"op34958(op11867)" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(0, 2), op46166(-0.000
%"op11334(op34958)" = fmul float 2.000000e+00, %"op34958(op11867)"
%"op16197(op11334)" = fadd float -1.000000e+00, %"op11334(op34958)"
%"op10954(op16197, op16197)" = fmul float %"op16197(op11334)", %"op16197(op11334)"
%"op34958(op11867)169" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(0, 2), op46166(-0.
%"op11334(op34958)170" = fmul float 2.000000e+00, %"op34958(op11867)169"
%"op16197(op11334)171" = fadd float -1.000000e+00, %"op11334(op34958)170"
%"op10954(op16197, op16197)172" = fmul float %"op16197(op11334)171", %"op16197(op11334)171"
%"op14323(op10954, op10954)" = fadd float %"op10954(op16197, op16197)", %"op10954(op16197, op16197)172"
%"op34958(op11867)173" = extractelement <4 x float> %"op11867gfx_sample(vl20713gfx_input(0, 2), op46166(-0.
%"op11334(op34958)174" = fmul float 2.000000e+00, %"op34958(op11867)173"
%"op16197(op11334)175" = fadd float -1.000000e+00, %"op11334(op34958)174"
%"op10954(op16197, op16197)176" = fmul float %"op16197(op11334)175", %"op16197(op11334)175"
%"op13436(op10954, op14323)" = fadd float %"op10954(op16197, op16197)176", %"op14323(op10954, op10954)"
%"op67044(op13436)" = call float @llvm.sqrt.f32(float %"op13436(op10954, op14323)")
%"op10439(op67044)" = fdiv float 1.000000e+00, %"op67044(op13436)"
%"op13008(op10439, op16197)" = fmul float %"op10439(op67044)", %"op16197(op11334)"
%"vl16927gfx_input(5, 2)" = call float @gfx_input(i32 5, i32 2)
%"op16911(vl16927, vl16927)" = fmul float %"vl16927gfx_input(5, 2)", %"vl16927gfx_input(5, 2)"
%"vl24069gfx_input(6, 2)" = call float @gfx_input(i32 6, i32 2)
%"op16911(vl24069, vl24069)" = fmul float %"vl24069gfx_input(6, 2)", %"vl24069gfx_input(6, 2)"
%"op14323(op16911, op16911)" = fadd float %"op16911(vl16927, vl16927)", %"op16911(vl24069, vl24069)"
%"vl29364gfx_input(7, 2)" = call float @gfx_input(i32 7, i32 2)
%"op16911(vl29364, vl29364)" = fmul float %"vl29364gfx_input(7, 2)", %"vl29364gfx_input(7, 2)"
%"op13436(op14323, op16911)" = fadd float %"op14323(op16911, op16911)", %"op16911(vl29364, vl29364)"
%"op67044(op13436)177" = call float @llvm.sqrt.f32(float %"op13436(op14323, op16911)")
%"op10439(op67044)178" = fdiv float 1.000000e+00, %"op67044(op13436)177"
%"op11513(op10439, vl16927)" = fmul float %"op10439(op67044)178", %"vl16927gfx_input(5, 2)"
%"op13578(op11513, op13008)" = fmul float %"op11513(op10439, vl16927)", %"op13008(op10439, op16197)"
%"op13008(op10439, op16197)179" = fmul float %"op10439(op67044)", %"op16197(op11334)171"
%"op11513(op10439, vl24069)" = fmul float %"op10439(op67044)178", %"vl24069gfx_input(6, 2)"
%"op13578(op11513, op13008)180" = fmul float %"op11513(op10439, vl24069)", %"op13008(op10439, op16197)179"
%"op14323(op13578, op13578)" = fadd float %"op13578(op11513, op13008)", %"op13578(op11513, op13008)180"
%"op13008(op10439, op16197)181" = fmul float %"op10439(op67044)", %"op16197(op11334)175"
%"op11513(op10439, vl29364)" = fmul float %"op10439(op67044)178", %"vl29364gfx_input(7, 2)"
%"op13578(op11513, op13008)182" = fmul float %"op11513(op10439, vl29364)", %"op13008(op10439, op16197)181"
%"op13436(op13578, op14323)" = fadd float %"op13578(op11513, op13008)182", %"op14323(op13578, op13578)"
%"vl13313gfx_input(8, 2)" = call float @gfx_input(i32 8, i32 2)
%"op16911(vl13313, vl13313)" = fmul float %"vl13313gfx_input(8, 2)", %"vl13313gfx_input(8, 2)"
%"vl33236gfx_input(9, 2)" = call float @gfx_input(i32 9, i32 2)
%"op16911(vl33236, vl33236)" = fmul float %"vl33236gfx_input(9, 2)", %"vl33236gfx_input(9, 2)"
%"op14323(op16911, op16911)183" = fadd float %"op16911(vl13313, vl13313)", %"op16911(vl33236, vl33236)"
%"vl40642gfx_input(10, 2)" = call float @gfx_input(i32 10, i32 2)
%"op16911(vl40642, vl40642)" = fmul float %"vl40642gfx_input(10, 2)", %"vl40642gfx_input(10, 2)"
%"op13436(op14323, op16911)184" = fadd float %"op14323(op16911, op16911)183", %"op16911(vl40642, vl40642)"
%"op67044(op13436)185" = call float @llvm.sqrt.f32(float %"op13436(op14323, op16911)184")
%"op10439(op67044)186" = fdiv float 1.000000e+00, %"op67044(op13436)185"
%"op11513(op10439, vl13313)" = fmul float %"op10439(op67044)186", %"vl13313gfx_input(8, 2)"
%"op11624(op11513)" = fsub float -0.000000e+00, %"op11513(op10439, vl16927)"
1: 1
Default text
1: 1
Default text
; ModuleID = 'PS_1.ll'
; ModuleID = 'PS_2.ll'
25 difference section(s)
Important Difference
Insert
Load time: 0.15 seconds

Other little things…

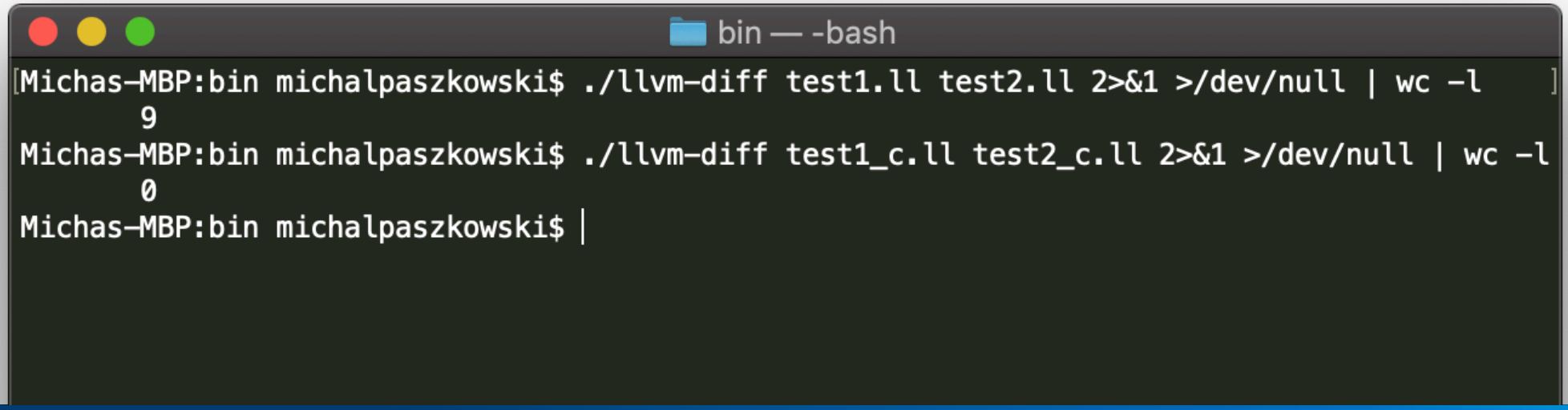
- Naming basic blocks
- Numbering function arguments
- Sorting values in PHI nodes

llvm-canon vs llvm-diff

Why not integrate with llvm-diff?

- We wanted to use this tool to also spot differences in just one file.
- We wanted to leverage existing diff tools.

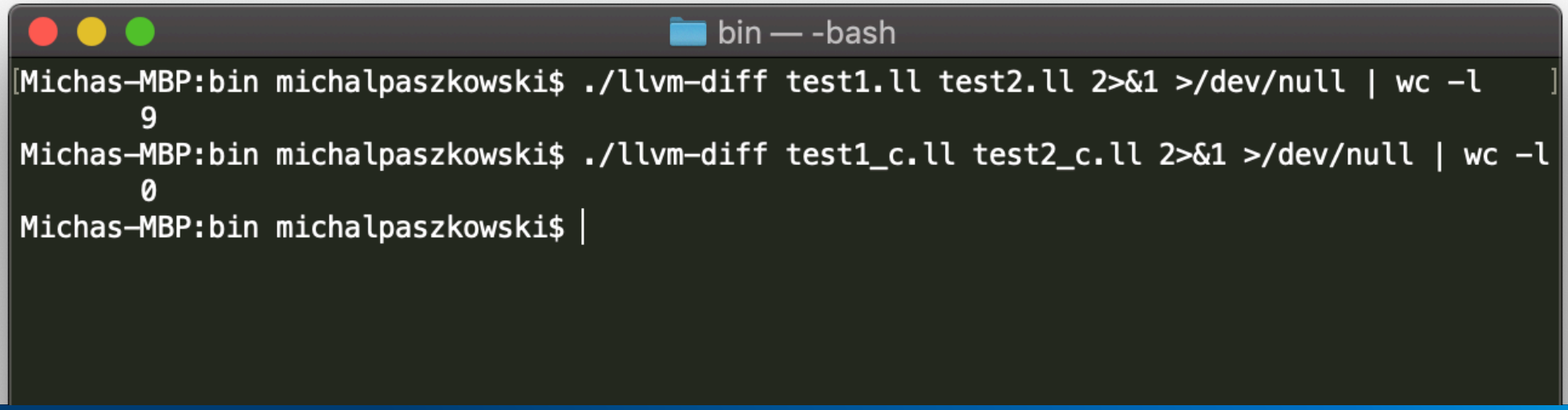
However, llvm-canon could be used as a prepass before using llvm-diff:



llvm-canon vs llvm-diff

```
define double @foo(double %a0, double %a1) {
entry:
  %a = fmul double %a0, %a1
  %b = fmul double %a0, 2.000000e+00
  %c = fmul double %a, 6.000000e+00
  %d = fmul double %b, 6.000000e+00
  ret double %d
}
```

```
define double @foo(double %a0, double %a1) {
entry:
  %a = fmul double %a0, %a1
  %c = fmul double 6.000000e+00, %a
  %b = fmul double %a0, 2.000000e+00
  %d = fmul double 6.000000e+00, %b
  ret double %d
}
```



Special thanks

Thanks to Puyan Lotfi for his talk on MIR-Canon during 2018 EuroLLVM and a head start on canonicalization techniques.

Special thanks to Radoslaw Drabinski and Julia Koval.

I would like to also thank the LLVM community for excellent code review and coming to this talk.

Q&A