MIR Patterns for GloballSel Combiners

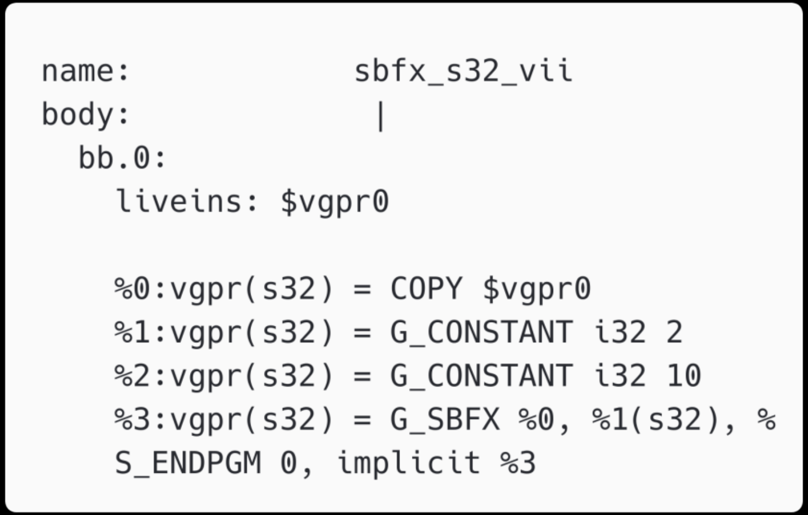
Pierre van Houtryve

AMD

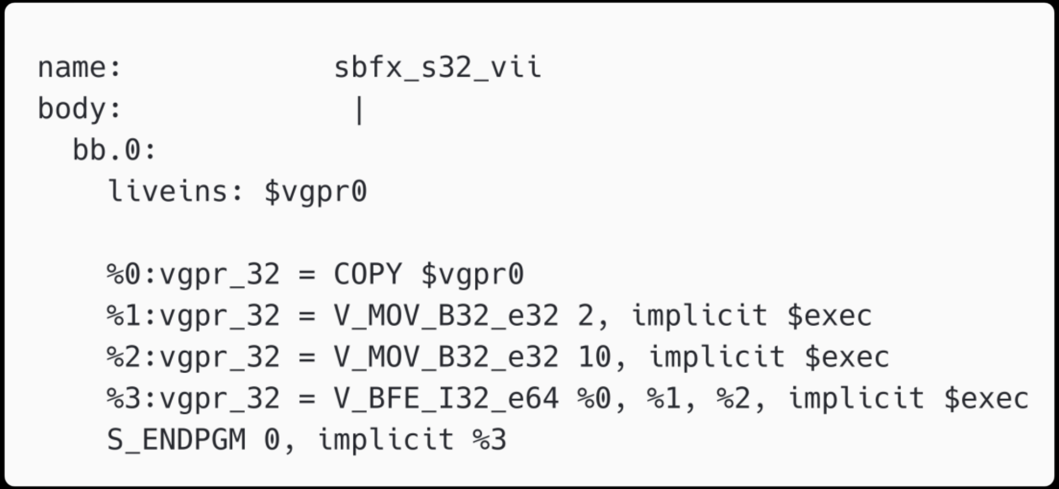
together we advance_

GlobalISel

- DAGISel alternative
- Uses (g)MIR
- Function scope



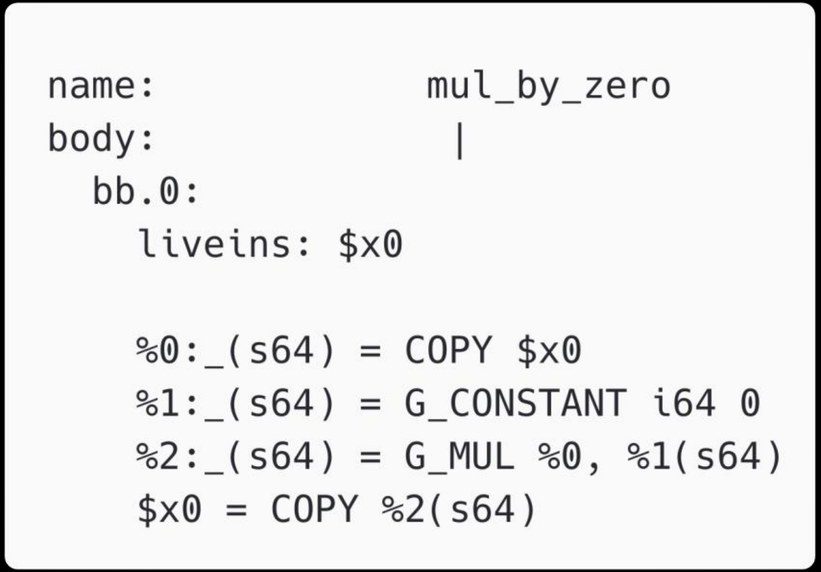
```
name:            sbfx_s32_vii
body:             |
  bb.0:
    liveins: $vgpr0

    %0:vgpr(s32) = COPY $vgpr0
    %1:vgpr(s32) = G_CONSTANT i32 2
    %2:vgpr(s32) = G_CONSTANT i32 10
    %3:vgpr(s32) = G_SBFX %0, %1(s32), %
    S_ENDPGM 0, implicit %3
```



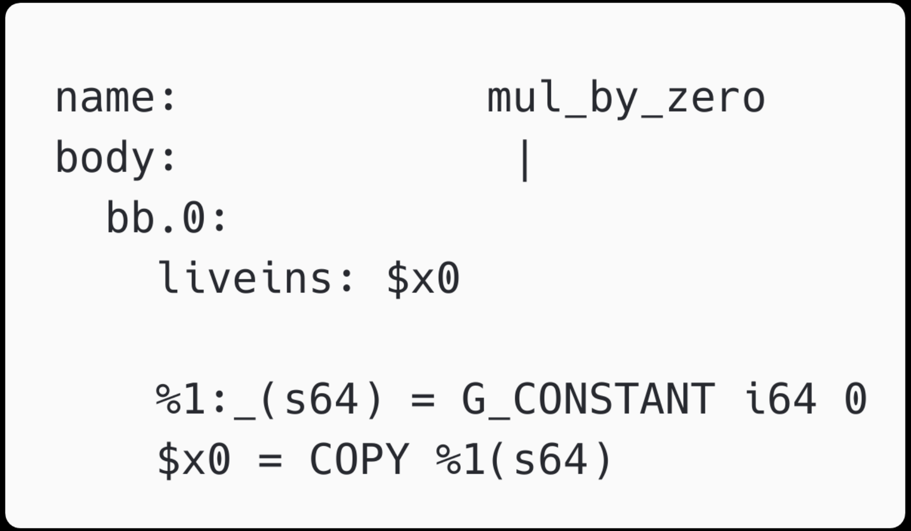
```
name:            sbfx_s32_vii
body:             |
  bb.0:
    liveins: $vgpr0

    %0:vgpr_32 = COPY $vgpr0
    %1:vgpr_32 = V_MOV_B32_e32 2, implicit $exec
    %2:vgpr_32 = V_MOV_B32_e32 10, implicit $exec
    %3:vgpr_32 = V_BFE_I32_e64 %0, %1, %2, implicit $exec
    S_ENDPGM 0, implicit %3
```

GlobalISel Combiners

- Matches and rewrites code patterns
- Generic rules & target-specific rules



```
name:             mul_by_zero
body:              |
  bb.0:
    liveins: $x0

    %0:_(s64) = COPY $x0
    %1:_(s64) = G_CONSTANT i64 0
    %2:_(s64) = G_MUL %0, %1(s64)
    $x0 = COPY %2(s64)
```



```
name:             mul_by_zero
body:              |
  bb.0:
    liveins: $x0

    %1:_(s64) = G_CONSTANT i64 0
    $x0 = COPY %1(s64)
```

GlobalISel Combiners: Input Before

```
// Fold (fabs (fneg x)) -> (fabs x).
def fabs_fneg_fold: GICombineRule <
  (defs root:$root, build_fn_matchinfo:$matchinfo),
  (match (wip_match_opcode G_FABS):$root,
         [{ return Helper.matchCombineFAbsOfFNeg(*${root}, ${matchinfo}); }]),
  (apply [{ Helper.applyBuildFnNoErase(*${root}, ${matchinfo}); }])>;
```

GlobalISel Combiners: Input Before

```
// idempotent operations
// Fold (freeze (freeze x)) -> (freeze x).
// Fold (fabs (fabs x)) -> (fabs x).
// Fold (fcanonicalize (fcanonicalize x)) -> (fcanonicalize x).
def idempotent_prop : GICombineRule<
   (defs root:$mi),
   (match (wip_match_opcode G_FREEZE, G_FABS, G_FCANONICALIZE):$mi,
          [{ return MRI.getVRegDef(${mi}->getOperand(1).getReg())->getOpcode() ==
                    ${mi}->getOpcode(); }]),
   (apply [{ Helper.replaceSingleDefInstWithOperand(*${mi}, 1); }])>;
```

GlobalISel Combiners: Output Before

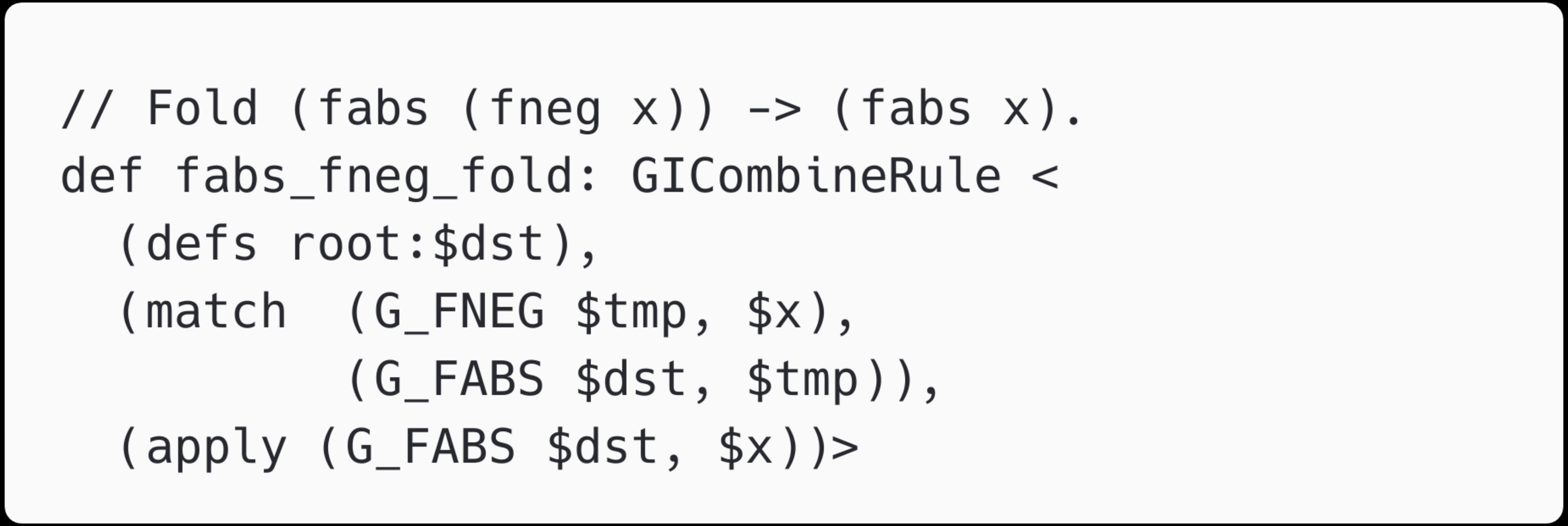
```
if (Partition == 4 /* TargetOpcode::G_FREEZE */) {
  // Leaf name: idempotent_prop
  // Rule: idempotent_prop
  if (!RuleConfig->isRuleDisabled(4)) {
    if (1
        && [&]() {
         return MRI.getVRegDef(MIs[0]->getOperand(1).getReg())->getOpcode() == MIs[0]->getOpcode();
         return true;
            }()
       ) {
      LLVM_DEBUG(dbgs() << "Applying rule 'idempotent_prop'\n");
      Helper.replaceSingleDefInstWithOperand(*MIs[0], 1);
      return true;
    }
  }
  return false;
}
```

AMD
together we advance_

GlobalISel Combiners: Refactoring Goals

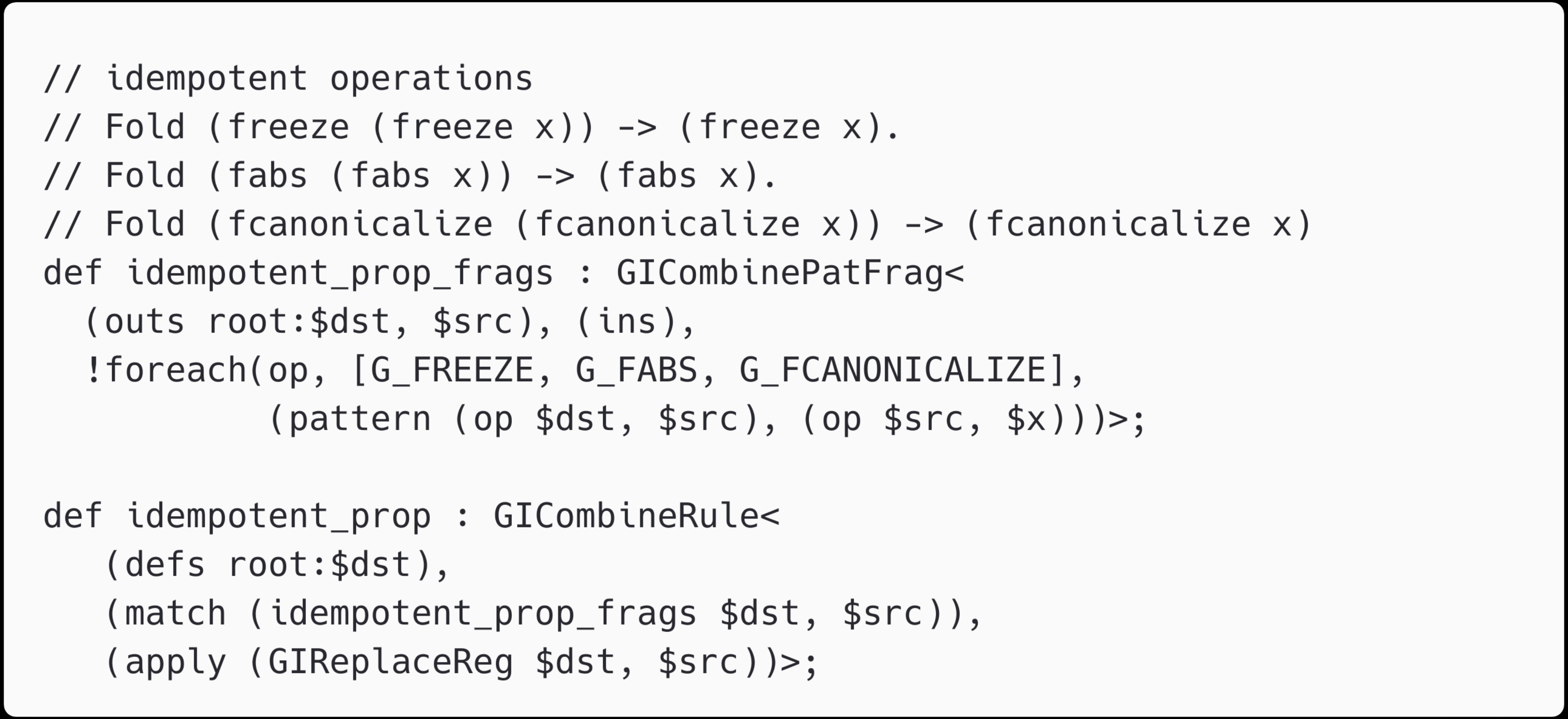
- Unify InstructionSelector and combiners infrastructure
- Allow doing more in pure TableGen (e.g., rewriting patterns)

GlobalISel Combiners: Input After



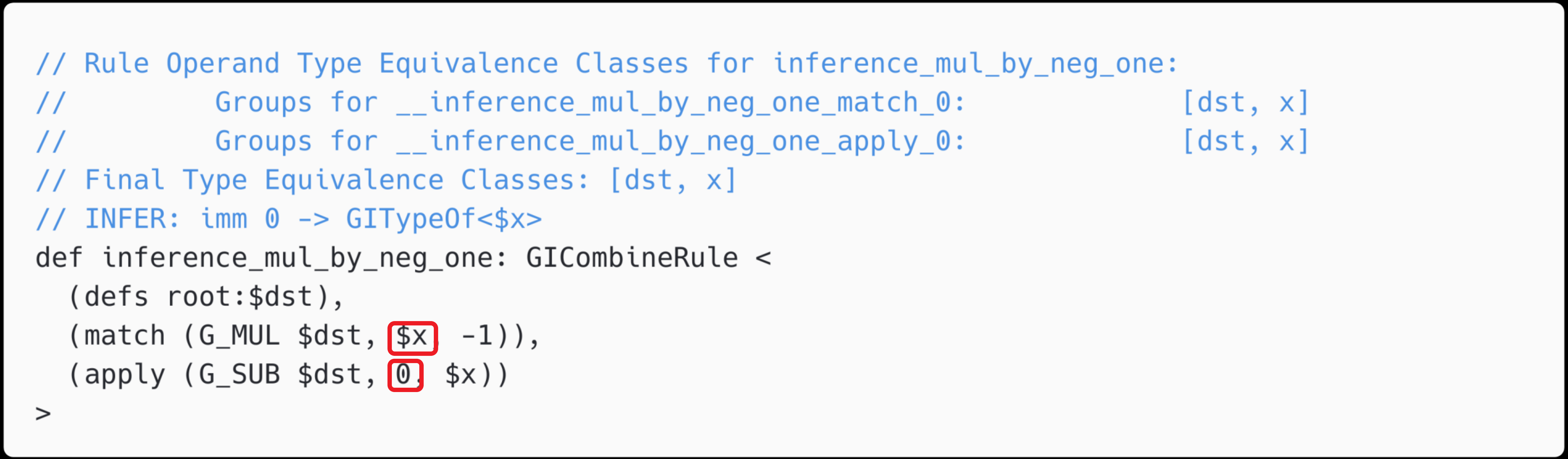
```
// Fold (fabs (fneg x)) -> (fabs x).
def fabs_fneg_fold: GICombineRule <
  (defs root:$dst),
  (match  (G_FNEG $tmp, $x),
          (G_FABS $dst, $tmp)),
  (apply (G_FABS $dst, $x))>
```

GlobalISel Combiners: Input After



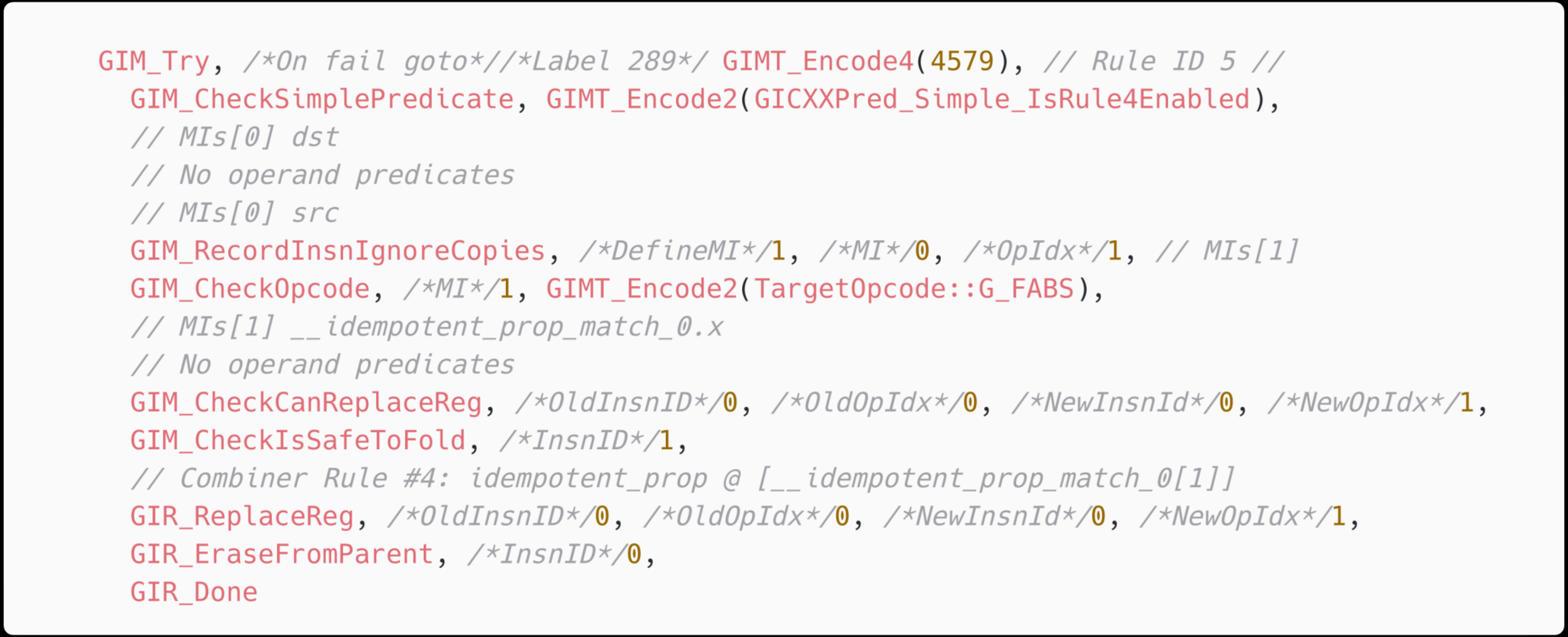
```
// idempotent operations
// Fold (freeze (freeze x)) -> (freeze x).
// Fold (fabs (fabs x)) -> (fabs x).
// Fold (fcanonicalize (fcanonicalize x)) -> (fcanonicalize x)
def idempotent_prop_frags : GICombinePatFrag<
  (outs root:$dst, $src), (ins),
  !foreach(op, [G_FREEZE, G_FABS, G_FCANONICALIZE],
           (pattern (op $dst, $src), (op $src, $x)))>;

def idempotent_prop : GICombineRule<
   (defs root:$dst),
   (match (idempotent_prop_frags $dst, $src)),
   (apply (GIReplaceReg $dst, $src))>;
```

GlobalISel Combiners: Type Inference



```
// Rule Operand Type Equivalence Classes for inference_mul_by_neg_one:
//         Groups for __inference_mul_by_neg_one_match_0:             [dst, x]
//         Groups for __inference_mul_by_neg_one_apply_0:             [dst, x]
// Final Type Equivalence Classes: [dst, x]
// INFER: imm 0 -> GITypeOf<$x>
def inference_mul_by_neg_one: GICombineRule <
  (defs root:$dst),
  (match (G_MUL $dst, $x, -1)),
  (apply (G_SUB $dst, 0, $x))
>
```

GlobalISel Combiners: Output After



```
GIM_Try, /*On fail goto*//*Label 289*/ GIMT_Encode4(4579), // Rule ID 5 //
  GIM_CheckSimplePredicate, GIMT_Encode2(GICXXPred_Simple_IsRule4Enabled),
  // MIs[0] dst
  // No operand predicates
  // MIs[0] src
  GIM_RecordInsnIgnoreCopies, /*DefineMI*/1, /*MI*/0, /*OpIdx*/1, // MIs[1]
  GIM_CheckOpcode, /*MI*/1, GIMT_Encode2(TargetOpcode::G_FABS),
  // MIs[1] __idempotent_prop_match_0.x
  // No operand predicates
  GIM_CheckCanReplaceReg, /*OldInsnID*/0, /*OldOpIdx*/0, /*NewInsnId*/0, /*NewOpIdx*/1,
  GIM_CheckIsSafeToFold, /*InsnID*/1,
  // Combiner Rule #4: idempotent_prop @ [__idempotent_prop_match_0[1]]
  GIR_ReplaceReg, /*OldInsnID*/0, /*OldOpIdx*/0, /*NewInsnId*/0, /*NewOpIdx*/1,
  GIR_EraseFromParent, /*InsnID*/0,
  GIR_Done
```

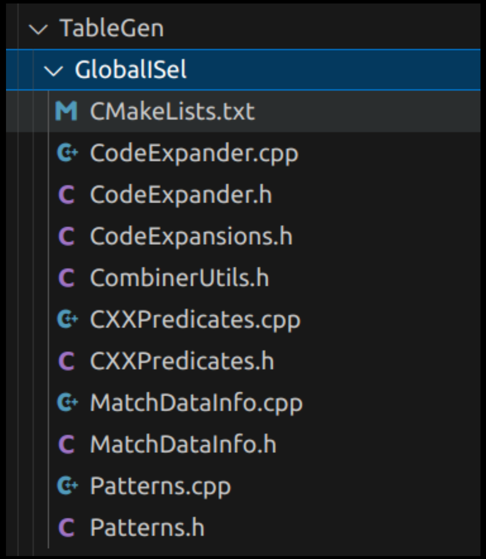
GlobalISel Combiners: Error Handling

- "Assert is an error" -> Diagnose errors, assert is a bug
    - Every diagnostic is tested

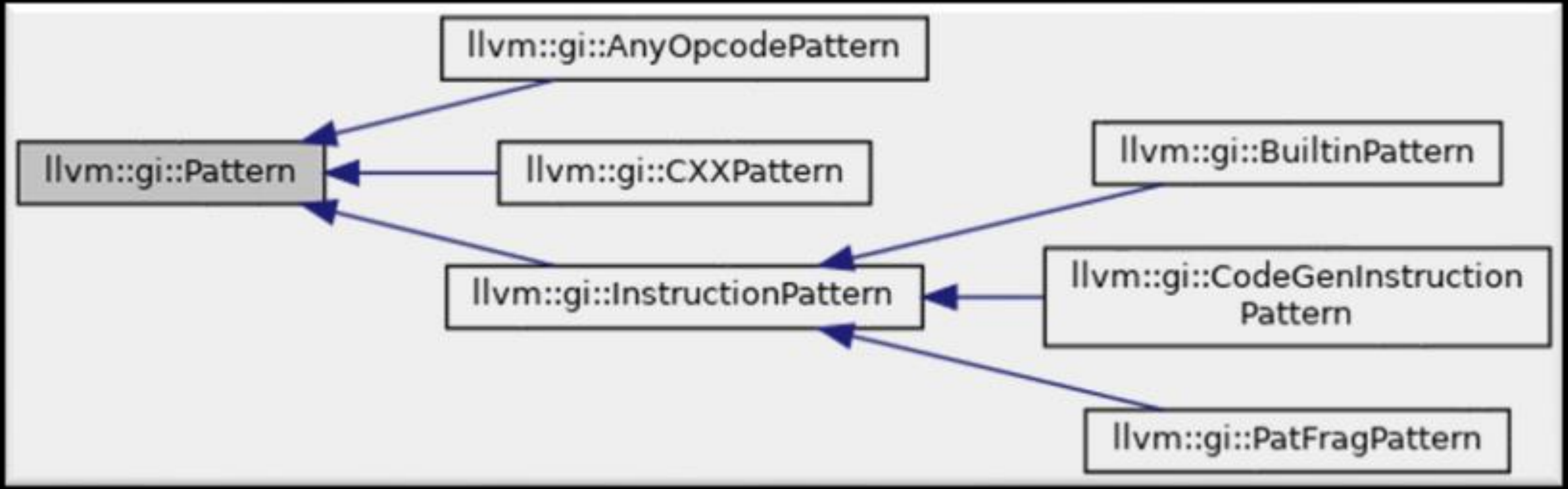
```
error: invalid output operand 'x': operand is not a live-in of the match pattern, and it has no definition

error: pattern 'foo' ('COPY') is unreachable from the pattern root!

warning: impossible type constraints: operand 1 of 'broken' has type 'i64', but 'TypedParams' constrains it
to 'i32'
note: operand 1 of 'broken' is 'k'
note: argument 1 of 'TypedParams' is 'i'
```

GlobalISel Combiners: Backend Design

- Good test coverage!
- Designed with reusability in mind (to some extent)



```
TableGen
  GlobalISel
    CMakeLists.txt
    CodeExpander.cpp
    CodeExpander.h
    CodeExpansions.h
    CombinerUtils.h
    CXXPredicates.cpp
    CXXPredicates.h
    MatchDataInfo.cpp
    MatchDataInfo.h
    Patterns.cpp
    Patterns.h
```


llvm::gi::AnyOpcodePattern
llvm::gi::Pattern
llvm::gi::CXXPattern
llvm::gi::BuiltinPattern
llvm::gi::InstructionPattern
llvm::gi::CodeGenInstruction Pattern
llvm::gi::PatFragPattern

GlobalISel Combiners: Limitations

- MIR patterns are (currently) only for simple patterns
  - Many rules still need a blend of C++ and MIR patterns, or full C++
- MIR patterns cannot...
  - Use KnownBits
  - Constraint constants (e.g., K is a multiple of 2)
  - Express constraints on types other than equality (e.g., T is 32 bits or lower)
  - Recursively match something
  - Etc.

GlobalISel MIR Patterns: What Now?

- Patterns become increasingly difficult to port
  - Effort >>> Reward
  - Feel free to request features by opening an issue
- Should we try using MIR patterns for ISel?
  - Interested? Come talk!
- DAG Syntax can be limiting
  - Should we consider parsing MIR directly at some point?

Do you like the concept of MIR patterns and have ideas?

Let's discuss!

Copyright and disclaimer

-
-

- The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

- THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION

AMD