Memcheck Reloaded: dealing with compiler-generated branches on undefined values

Julian Seward, jseward@acm.org

2 February 2020. FOSDEM. Brussels.

Motivation

Memcheck checks

- Whether memory accceses are to allowable locations
  - (Relatively) easy
- Whether branches depend on undefined values
  - (Relatively) difficult
- Low false positive rates are very important
  - Circa 2005   Everything under control
  - Circa 2015   Increasingly problematic – clang 3+, gcc 5+
- Overview
  - Some definedness tracking examples
  - The undefined-conditional-branch problem
  - The solution

Some basics

For every bit of process state, Memcheck maintains a shadow (“V”) bit
all registers and memory locations are shadowed

`1` means Undefined. `0` means Defined.

When program computes a result from operands ..

```
r = x + y
```

.. Memcheck computes definedness of result from definedness of operands

```
r# = ... x# ... y# ...
```

When program does a conditional branch, Memcheck checks definedness of the condition
and emits an error if undefined

As described in our Usenix 2004 paper (Seward & Nethercote)

`http://valgrind.org/docs/memcheck2005.pdf`

Tracking definedness in value flows

In principle ..

We know exact definedness behaviour of `AND`, `OR`, `NOT`

`NOT: 0 → 1,  1 → 0,  U → U`

`AND: (0,U) → 0,   (1,U) → U,` for non-`U` inputs as expected

Any arithmetic op can be reduced to an `AND/OR/NOT` formula

→ we can derive “exact” definedness propagation for any op

In practice ..

Way too expensive

Use cheap approximations

Mostly OK – undef value use hard to reason about

Value flows #2

Eg Integer Add

Simplest: all output bits are `U` if any input bit is `U`

```
   10U0 +# 0001  →  UUUU
```

Too crude .. can’t deal with “overwidth” adds

Better: we know undefinedness propagates only leftwards

```
   10U0 +# 0001  →  UUU1
```

Best: defined zeroes stop leftward propagation

```
   10U0 +# 0001  →  10U1
```

Costs: circa, 3, 5, 10 insns

Value flows #3

Choose approximations from real-world experience

Add/Sub: inexact ("Better") for address computations
exact ("Best") for everything else

And/Or: exact: AND with 0, OR with 1 → Defined

Integer ==: defined 1 vs defined 0 makes result Defined
even if all other bits undefined

Shifts tracked exactly

Most other ops approximated safely – input undefinedness pollutes entire output

Things it **doesn't** know, eg:

Undefined * zero → Defined

`x >=unsigned 101010000` → Defined even if lowest 4 bits of `x` are undefined

And this worked pretty well. Until ...

The Problem

… until .. complaints on this

```
   int result
   bool ok = compute_something(&result)
   if (ok && result == 42) { … }         ←------------- ERROR REPORTED!
```

Why? ‘cos clang/gcc compiled it like this:

```
   if (result == 42 && ok) { … }
```

Compiler’s buggy. Right?

well actually

`A && B  ==  B && A`   if `A` is false whenever `B` is undefined

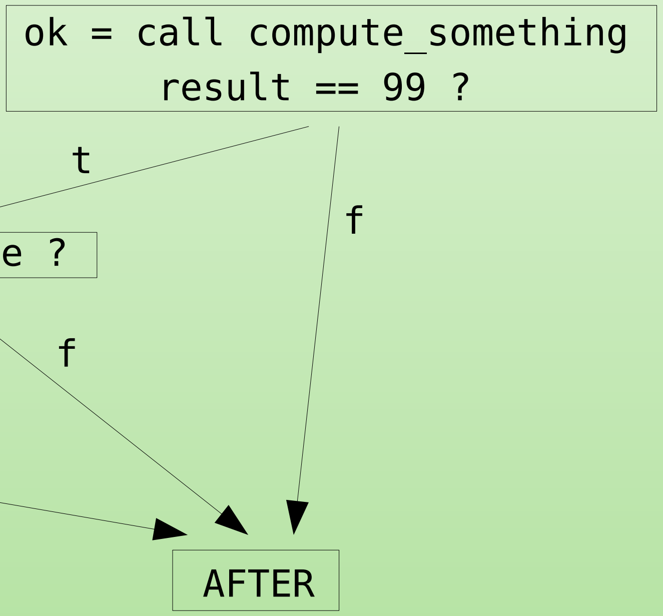
Program and compiler are correct, so why is this error reported?

Why is this a problem

Unit of analysis is “basic block”

- Straight line code ending in branch
- Memcheck assumes every conditional branch is “important”

```
int result
bool ok = compute_something(&result)
if (result == 99 && ok) { … STUFF .. }
.. AFTER ..
```



Why is this a problem #2

Can’t “see” over multiple blocks

Basic-blockness deeply wired in

What to do?

- Complex
- ”If an undef value use is observed in a conditional branch, only report it if the architected machine state is changed before we arrive at the instruction which is the immediate postdominator of the branch ...”
- .. or something like that.

Naaah

- Way too complex

What do to?

Summer 2018

Depressed. “End of the road for Memcheck”

Winter 2018

Depressed. (Cold and dark)

Summer 2019

Hmm. Didn’t we already solve this problem before?

Same as pure-value-flow for `AND`

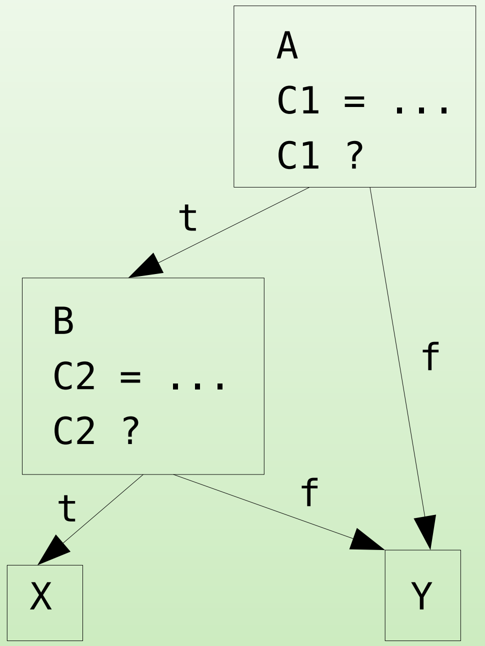
```
AND  0, Undefined  ==   AND Undefined, 0  ==  Defined-0
```

Problem is, the `AND` is spread over multiple blocks

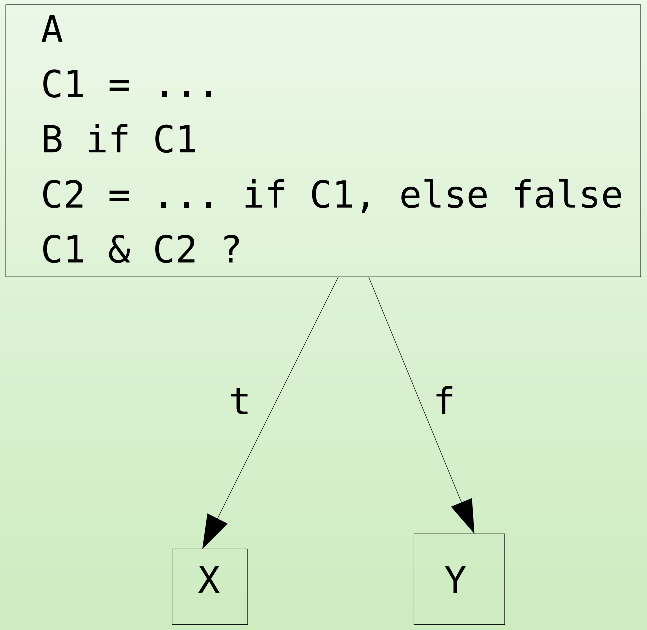
Need to “recover”/”reconstruct” it

So here's the plan

Transform this ...



into ... this



Be careful about `B` when `C1` is `false`

Now we can use value-level exact instrumentation for &

Implementation feasibility

Don’t want to do this per-arch (ARM, x86, Power, Mips, S390)

- But their branch insns are all different
- Leverage Valgrind’s IR infrastructure
    - Translate to IR
    - normalise (a.k.a ”optimise”)
    - pattern-match
    - transform

This will slow down the JIT

- True. But not much
- Backend costs dominate
- This is front-end

Same mechanism handles source level && and ||

So, in conclusion ..

Memcheck lives to ride another day   \o/ \o/ \o/

Can run Firefox compiled with clang -O2, gcc -O2, with zero false positives (of this kind)

Available on x86 32/64, arm 32/64, power 32/64
  MIPS 32/64 and S390 crash for unknown reasons
  On those targets, is disabled until it can be fixed

In the tree now; seems stable

Will be in Valgrind 3.16

Thank you for listening!

Questions?