Integer overflow inside Event-B action formulas

Fredrik Öhrström (fredrik.ohrstrom@viklauverk.com)
30 may 2023

Background

I created the EVBT tool to generate documentation and code for machines developed using Rodin.

`https://github.com/viklauverk/EventBTool`

From a Rodin project it can generate documentation

```
EVENT ImproveUpperBound
REFINES ImproveUpperBound
WHERE
  grd4_1:  low + 1 ≠ high
  grd4_2:  mid * mid > input
WITH
  m:  m = mid   mid is a better value for high
THEN
  act4_1:  high := mid
  act4_2:  mid := (low + mid) ÷ 2
END
```

And it can generate code

```
bool SquareRootImplementation::ImproveUpperBound()
{
    bool grd4_1 = low + 1 != high; // low+1≠high
    if (!grd4_1) return false;
    bool grd4_2 = mid * mid > input; // mid∗mid>input
    if (!grd4_2) return false;
    high = mid; // high ≔ mid
    mid = (low + mid) / 2; // mid ≔ (low+mid)÷2
    traceEvent("ImproveUpperBound");
    return true;
}
```

And it can generate code

```
int SquareRootImplementation::run()
{
    int c = 0;
    while (true) {
        if (ImproveLowerBound()) { c++; continue; }
        if (ImproveUpperBound()) { c++; continue; }
        if (SquareRoot()) { c++; continue; }
        // No event has triggered, the machine has stopped.
        break;
    }
    return c;
}
```

The big question is though...

Why does it work?

The big question is though...

Mostly because of implicit widening/narrowing of integers in the C-compiler.

Overflow when implicit widening stops at a byte

```
machine Overflow

variables x y

invariants
  @inv1 x∈0‥255
  @inv2 y∈0‥255

events

  event MERGA
    where
      @grd1 y > 0
      @grd2 y > x
    then
      @act1 x≔(x∗x)÷y
  end
```

Overflow when implicit widening stops at a byte

```
x = 0x80

y = 0x88

0x80 * 0x80 / 0x88 = 0x4000 / 0x88 = 0x78
```

but with overflow.

```
0x80 * 0x80 / 0x88 = 0x00 / 0x88 = 0x00
```

Overflow when implicit widening stops at an int

```
x = 0x40000000

y = 0x40000040
```

```c
int main()
{
    int x = 0x40000000;
    int y = 0x40000040;

    int sum = (x*x)/y;

    printf("(%d * %d) / %d = %d %x\n", x, x, y, sum, sum);
}
```

```
(1073741824 * 1073741824) / 1073741888 = 0 0
```

Overflow with signed ints is even worse

Overflowing of signed values is undefined!

I have experienced that in a g++ generated loop, adding 2 to 2147483647 (0x7fffffff) the result will sometimes wrap to negative, but sometimes it stays put at the same value!

This behaviour is permitted since signed integer overflow is undefined. You can make it stable with -fwrapv which forces a defined overflow behaviour.

You should run debug with -fsanitize=undefined to catch such bad runtime behaviour in an explicit error.

Only way to prevent widening

```
x += 5;
```

Or we can build a C++ class to protect us from widening:

```
U16 u16_add_u8u8(U8 a, U8 b)
{
    uint16_t sum = a.v();
    sum += b.v();
    return U16(sum);
}
```

Only way to prevent narrowing

And to protect us from narrowing:

```
struct U8
{
    U8() : v_(0) {}
    U8(uint8_t v) : v_(v) {}
    uint8_t v() { return v_; }
    U8(int8_t v) = delete; // Prevent implicit narrowing!
    U8(uint16_t v) = delete; // Prevent implicit narrowing!
    U8(int16_t v) = delete; // Prevent implicit narrowing!
    U8(uint32_t v) = delete; // Prevent implicit narrowing!
    U8(int32_t v) = delete; // Prevent implicit narrowing!
    U8(uint64_t v) = delete; // Prevent implicit narrowing!
    U8(int64_t v) = delete; // Prevent implicit narrowing!
    U8(unsigned __int128 v) = delete; // Prevent implicit narrowing!
    U8(__int128 v) = delete; // Prevent implicit narrowing!


    private:
    uint8_t v_;
 };
```

Guards are also under suspicion

From the bridge traffic lights example.

EVENT ML_out1
REFINES ML_out1
WHERE

grd1: $ml\_out\_10 = \mathrm{TRUE}$
grd2: $a + b + 1 < \mathrm{d}$

THEN

act1: $a := a + 1$
act2: $ml\_pass := 1$
act3: $ml\_out\_10 := \mathrm{FALSE}$

END

What happens if d is defined as `MAX_INT`?

Guards are also under suspicion

```
bool BridgeTrafficLightsImplementation::ML_out1()
{
    bool grd1 = ml_out_10 == true; // ml_out_10=TRUE
    if (!grd1) return false;
    bool grd2 = a + b + 1 < d; // a+b+1<d
    if (!grd2) return false;
    a = a + 1; // a ≔ a+1
    ml_pass = 1; // ml_pass ≔ 1
    ml_out_10 = false; // ml_out_10 ≔ FALSE
    traceEvent("ML_out1");
    return true;
}
```

How should EVBT handle this…

Different strategies:

1. Delegate the problem to a human, that verifies that the microcontrollers MAX_INT (and the corresponding widening/narrowing) will be satisfactory.
2. Delegate to EVBT to determine the same.
3. Have EVBT generate code that explicitly widens (u8 to u16 to u32 etc) and generates code that works on these.
4. Have EVBT delegate the range checking to C++.
5. Have EVBT use BIGINTs from the start.
6. Have EVBT generate code that can switch from using int to BIGINT when needed.
7. Have EVBT detect overflow and terminate program.

Use EVBT to track necessary storage requirements

Promotion u8*u8 -> u16

Promotion u64*u64 -> BIGINT

But u8+u8 -> u9

Use EVBT to track necessary storage requirements

EVBT can store meta data inside the formula.

$$\mathrm{S} \nrightarrow \mathbb{P}(1..3)$$

$$\mathrm{S}\ll\mathrm{int8}\gg \nrightarrow \ll\mathrm{array}\gg\mathbb{P}\ll\mathrm{bitset8}\gg(1..3)$$

The purpose is to store the selected implementation details.

Use EVBT to track necessary storage requirements

But EVBT could also use the same meta-data to store the ranges.

$$height := (47 * delta)$$

$$height := \ll\text{safe}\mapsto\text{int8}\gg(47\ll\text{int8}\gg * \ll\text{int16}\gg delta\ll\text{int8}\gg)$$

Goals

1. The codegeneration should be able to deduce the necessary storage requirements for the formulas and guards.
2. For a not fully specified model that we want to run anyway, then EVBT can use BIGINTs or capping values to MAX_INT. However the runtime can then terminate exceptionally on OutOfMemory and/or Overflow.
3. If all the variables are bounded to reasonable values, then the generated code should have no runtime checks and be guaranteed to work because of the proven specification.

Conclusion and Questions

- The is tricky! Has this been solved already?
- Do I get these problems because I do not refine the model enough?
- Graceful handling of OutOfMemory/Overflow?
- The generated code will not be as easy to read.