The Whacky World of Undefined Behaviour

Beck, Calvin
hobbes@seas.upenn.edu

October 17, 2019

What is this Talk about?

Undefined behaviour! With a smack of LLVM.

We'll cover things like:

- What is undefined behaviour?
- What happens when you encounter UB?
- How is UB useful? Should we avoid it?
    - Optimizations?
- UB in LLVM (and indeterminate values)
- How this all fits into Vellvm

Not for anything in particular! It's just a fun topic, and hopefully talking about it will clarify some things for myself and you!

What is undefined behaviour (UB)?

It’s behaviour...

What is undefined behaviour (UB)?

It’s behaviour...

That’s undefined.

What is undefined behaviour (UB)?

It's behaviour...

That's undefined.

Done.

What is undefined behaviour (UB)?

Why does this seem hard?

What is undefined behaviour (UB)?

Why does this seem hard?

- Easy to conflate with things like implementation defined behaviour... Which is sort of different.

What is undefined behaviour (UB)?

Why does this seem hard?

- Easy to conflate with things like implementation defined behaviour... Which is sort of different.
- Language dependent.
  - Array out of bounds in Python? Exception, not UB.

What is undefined behaviour (UB)?

Why does this seem hard?

- Easy to conflate with things like implementation defined behaviour... Which is sort of different.
- Language dependent.
  - Array out of bounds in Python? Exception, not UB.
  - Array out of bounds in C? ... Pray.

What happens when you encounter UB?

What happens when you encounter UB?

ANYTHING.

What happens when you encounter UB?

ANYTHING.

Yes.

What happens when you encounter UB?

ANYTHING.

Yes. Anything.

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
  - No, seriously. Erase the hard drive.
  - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
  - No, seriously. Erase the hard drive.
  - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`
- time travel

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
    - No, seriously. Erase the hard drive.
    - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`
- time travel
    - no, really.
    - `https://devblogs.microsoft.com/oldnewthing/?p=633`

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
  - No, seriously. Erase the hard drive.
  - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`
- time travel
  - no, really.
  - `https://devblogs.microsoft.com/oldnewthing/?p=633`
- nasal demons?

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
  - No, seriously. Erase the hard drive.
  - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`
- time travel
  - no, really.
  - `https://devblogs.microsoft.com/oldnewthing/?p=633`
- nasal demons?
  - `https://en.wikipedia.org/wiki/Nasal_demons`

What happens when you encounter UB?

Compiler will do whatever it finds easiest or most efficient.

- noop, and then continue
- halt
- halt **and** catch fire
- erase the hard drive
  - No, seriously. Erase the hard drive.
  - `https://kristerw.blogspot.com/2017/09/why-undefined-behavior-may-call-never.html`
- time travel
  - no, really.
  - `https://devblogs.microsoft.com/oldnewthing/?p=633`
- nasal demons?
  - `https://en.wikipedia.org/wiki/Nasal_demons`
  - So far I'm pretty sure this is just a joke, but I wouldn't rule it out.

Why is this useful?

Why have UB at all? Isn't it... crazy?

Why is this useful?

Why have UB at all? Isn't it... crazy?

- PL without UB might have a lot of dynamic sanity checks.
  - Bounds checking.

Why is this useful?

Why have UB at all? Isn't it... crazy?

- PL without UB might have a lot of dynamic sanity checks.
  - Bounds checking.
- What about type systems?
  - Static checks can eliminate some dynamic checks
  - Bounds checking still common.

Instead, why not...

Why is this useful?

Why have UB at all? Isn't it... crazy?

- PL without UB might have a lot of dynamic sanity checks.
  - Bounds checking.
- What about type systems?
  - Static checks can eliminate some dynamic checks
  - Bounds checking still common.

Instead, why not...

Do nothing?

Why is this useful?

UB may seem somewhat unprincipled, but it has advantages:

- Gives compiler an axiom.
- Puts burden on programmer, or other tools

UB can reflect programmer intent

I want to change this...

```
a + b < a + c
```

UB can reflect programmer intent

I want to change this...

```
a + b < a + c
```

To this:

```
b < a
```

After all, who wants to do 2 extra additions?

UB can reflect programmer intent

I want to change this...

```
a + b < a + c
```

To this:

```
b < a
```

After all, who wants to do 2 extra additions?

But this is sort of wrong...

```
1 + INT_MAX < 1 + 3
// This evaluates to
INT_MIN < 4 == True

// But...
INT_MAX < 3 == False
```

Pointer aliasing — optimizations with undecidability

Undefined behaviour allows compilers to make optimizations based on undecidable invariants.

Pointer aliasing — optimizations with undecidability

Undefined behaviour allows compilers to make optimizations based on undecidable invariants.

It might be nice to optimize this:

```
void sum(double *total, double *array, size_t len )
{
    *total = 0;
    for (size_t i=0; i<len; i++) {
        *total += array[i];
    }
}
```

To this:

```
void sum(double *total, double *array, size_t len )
{
    double local_total = 0;
    for (size_t i=0; i<len; i++) {
        local_total += array[i];
    }

    *total = local_total;
}
```

C's restrict keyword

```
void sum(double* restrict total, double* restrict array, size_t len )
{
    *total = 0;
    for (size_t i=0; i<len; i++) {
        *total += array[i];
    }
}
```

Allows optimization to

```
void sum(double *total, double *array, size_t len )
{
    double local_total = 0;
    for (size_t i=0; i<len; i++) {
        local_total += array[i];
    }

    *total = local_total;
}
```

Because restrict says “these pointers don't alias with anything”. If they happen to alias, then it's UB, and the program can do whatever.

WHERE WE'RE GOING WE DON'T NEED ROADS

WHERE WE'RE GOING WE DON'T NEED ROADS

So, how powerful is undefined behaviour?

WHERE WE'RE GOING WE DON'T NEED ROADS

So, how powerful is undefined behaviour?

Can we optimize this:

```c
char inp = getchar();
if ('A' == inp) {
    printf("Hello, world!\n");
    x = 1 / 0;
}
```

WHERE WE'RE GOING WE DON'T NEED ROADS

So, how powerful is undefined behaviour?

Can we optimize this:

```
char inp = getchar();
if ('A' == inp) {
    printf("Hello, world!\n");
    x = 1 / 0;
}
```

To this?

```
getchar();
```

References

- In: ().
- .
- .

These are all good resources! You should look at them!