



Writing Software That's Safe  
Enough To Drive A Car

*@shnewto*

**Clickbait!**

# Functional safety is...

the absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems.

“State of the Art”

What is safe enough?



# Mutable aliasing

```
int a = 0;    // data
int *b = &a;  // alias
int **c = &b; // alias
```

```
*c += 2048; // "corruption" (allowed by static analysis tools)
*b = 1;    // crash
```



## Casting away the `const`

```
const int a = 0;  
*((int *)&a) = 1;
```

# Alternatively: We can't destroy what we don't own

```
let mut a: i32 = 0;  
let mut b: &mut i32 = &mut a;  
let c: &&mut i32 = &mut b;
```

```
drop(*c); // "corrupt"  
//~^ ERROR cannot move out of borrowed content
```

```
*b = 1;  
//~^ ERROR cannot assign to `*b` because it is borrowed
```



# Rust: Nope, still immutable

```
let a: i32 = 0;  
*(&mut a) = 1;  
//~^ ERROR cannot borrow immutable local variable `a` as mutable
```



# Pattern *mis*-matching

```
/* Enumeration intended for use. */
typedef enum { APPLY_BRAKE = 1, APPLY_THROTTLE = 2 } action_e;
/* Ambiguous enumeration that can results dangerous behavior. */
enum { DO_NOT_SELF_DESTRUCT = 1, SELF_DESTRUCT = 2, UNDECIDED = 3 };

action_e pattern = APPLY_THROTTLE;

switch (pattern)
{
    /* Destructive implications. */
    case DO_NOT_SELF_DESTRUCT: { break; }
    case SELF_DESTRUCT: { /*!!*/ break; }
    case UNDECIDED: { break; }
    default: { break; }
}
```

# Mismatch caught

```
enum Action { ApplyBrake = 1, ApplyThrottle = 2, }  
enum Destruct { SelfDestruct = 1, DoNotSelfDestruct = 2, Undecided = 3,}  
  
let pattern = Action::ApplyThrottle;  
  
match pattern {  
    Destruct::SelfDestruct => panic!("!!"), //~ ERROR mismatched types  
    Destruct::DoNotSelfDestruct => {} //~ ERROR mismatched types  
    Destruct::Undecided => {} //~ ERROR mismatched types  
}
```

If it doesn't compile, it can't crash.

C is proven in use, why change?

Redefining “State of the Art”

MISRA-Rust?

What's next?



# Resources

[github.com/PolySync/static-analysis-argumentation](https://github.com/PolySync/static-analysis-argumentation) (code)

[polysync.io/blog](https://polysync.io/blog)

- *The Challenge of Using C in Safety Critical Applications (white paper)*
- *Should Safety-Critical Software be Written in C? (blog post)*

[sheas.blog/talks](https://sheas.blog/talks) (slide deck)

Twitter: @shnewto

Blog: sheas.blog

GitHub: shnewto

Email: shnewto@gmail.com