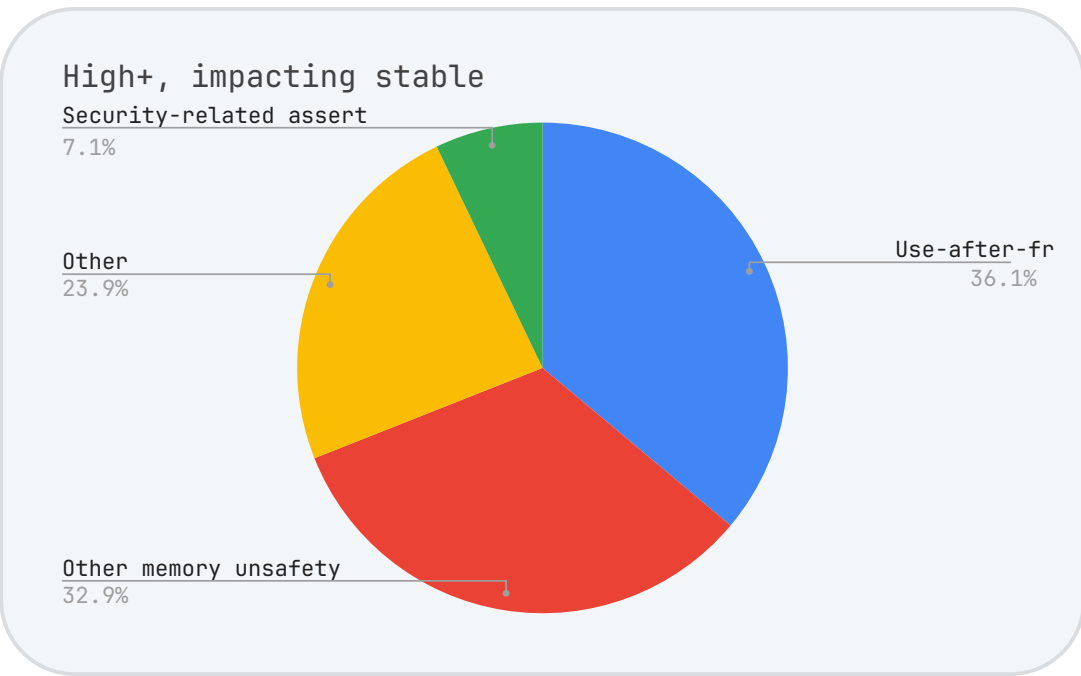


Oxidize: Macro-Aware C-to-Rust Translation

Robbe De Greef robbe.de.greef@vub.be
Antonio Paolillo antonio.paolillo@vub.be
SPELLS² – SOFT – Vrije Universiteit Brussel

1. Context

- Memory safety bugs account for roughly 70% of CVEs
- Can we rewrite legacy C code in a memory-safe language such as Rust?
- Can we do it automatically?



2. Problem

- “Flat” code
 - Inlined constants
 - Lost abstractions

```
1 void skipws(ZFILE *fp) {
2     int c;
3     while (c = Zgetc(fp), isascii(c) && isspace(c));
4     Zungetc(c, fp);
5 }
```

```
1 void skipws(ZFILE *fp)
2 {
3     int c;
4     while (c = ((--((fp)->len) >= 0) ? (unsigned char)*((fp)->ptr++) : _Zgetc(fp)),
5             (((c) & ~0x7f) == 0) && ((*__ctype_b_loc())[((int)((c))] &
6             (unsigned short int)_ISspace));
7     ((fp)->ptr--, (fp)->len++, (c));
8 }
```

3. Solutions

Macro contraction – Parse everything as an expression

```
1
2 #define LB {
3 #define BAR -1
4 int foo() LB
5     return BAR BAR;
6 }
```

```
1
2
3
4 int foo() {
5     return -1 -1;
6 }
```

```
1
2 macro_rules! BAR {
3     () => { -1 }
4 fn foo() -> i32 {
5     BAR!() - 1
6 }
```

Invoke expand algorithm – Recursively expand necessary macros

```
1 #define ONE 1
2 #define PLUS +
3 #define INC(a) a PLUS ONE
4 int foo() {
5     return INC(1);
6 }
```

```
1
2 #define ONE 1
3 #define INC(a) a + ONE
4 int foo() {
5     return INC(1);
6 }
```

Generic helper traits – Redefine operators with C semantics

```
1 #define ADD(a,b) a + b
2 void foo() {
3     int x = ADD((int)1, (char)2);
4     int *y = ADD(&x, 1);
5 }
```

```
1 macro_rules! ADD {
2     ($a:expr, $b:expr) =>
3     { $a.c_add($b) }
4 fn foo() {
5     let x = ADD!(1i32, 2i8);
6     let y = ADD!(&x, 1);
7 }
```

Passing in types – Types provided at invocation site

```
1 #define ASSIGN(l, r) l = r
2 void baz() {
3     int count;
4     ASSIGN(count, (char)1);
5 }
```

```
1 macro_rules! ASSIGN {
2     ($l:expr, $r:expr, $t0:ty) =>
3     { $l = $r as $t0 }
4 fn baz() {
5     let mut count: i32;
6     ASSIGN!(count, 1i8, i32);
7 }
```

Hygiene

- Check for identifier shadowing

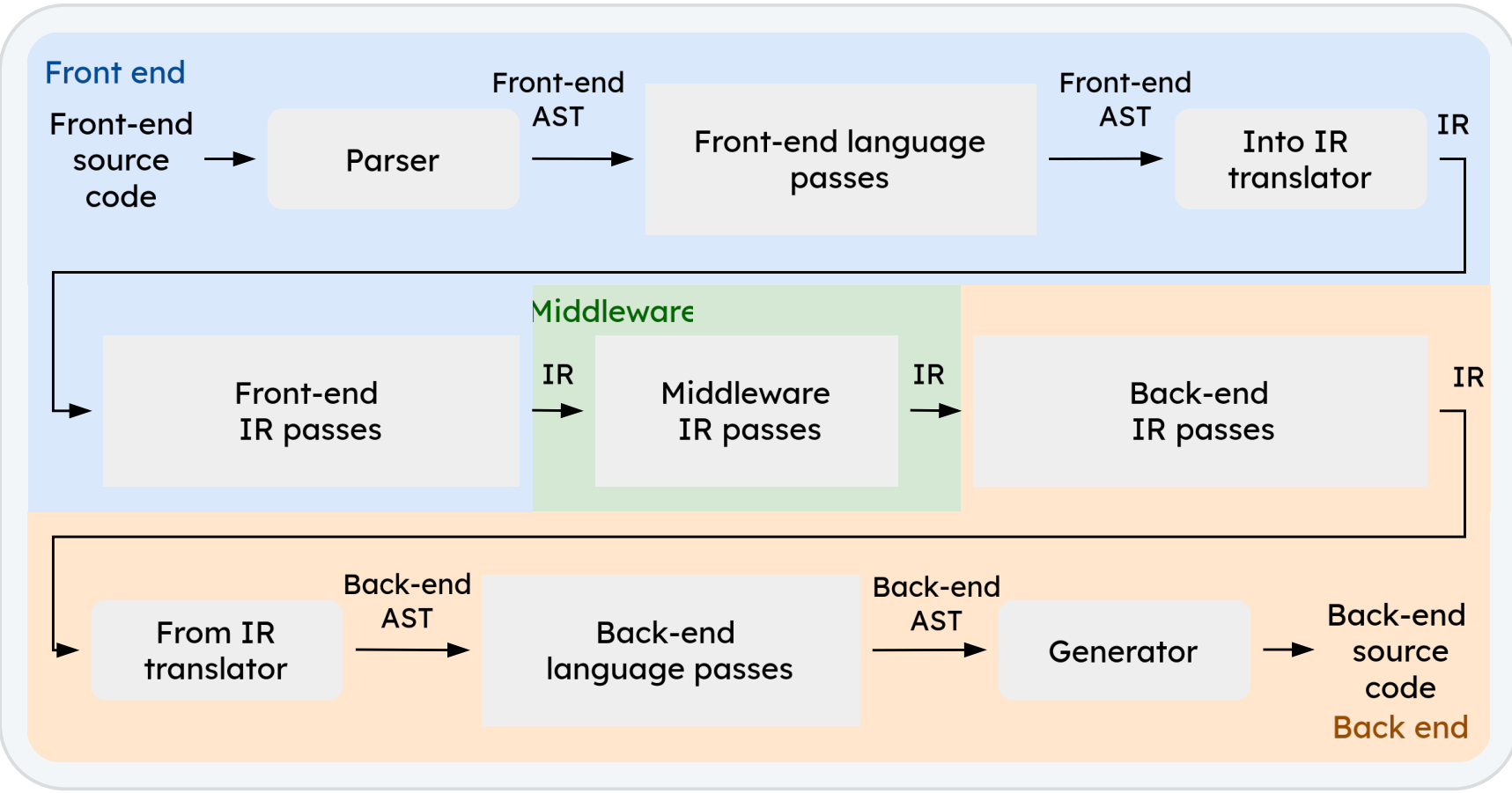
```
1 int count = 0;
2 #define COUNT count
```

```
1 #include <counter.h>
2 int foo() {
3     int count = bar();
4     return COUNT;
5 }
```

4. Oxidized example from §2

```
1 pub unsafe extern "C" fn skipws(mut fp: *mut ZFILE) -> () {
2     let mut c: i32 = (0 as i32);
3     while ({c = Zgetc!(fp, ...) as i32}; (((isascii!(c, ...) != 0) && (isspace!(c, ...) != 0)) as i32) != 0) {}
4     Zungetc!(c, fp, ...);
5 }
```

5. Oxidize architecture



- Inspired by MLIR and the Nanopass framework
- 28 frontend passes and 13 backend passes

6. Evaluation

Codebase coverage

Macro retention

	0	20	40	60	80	100
doom	46/84 (55%)	76/84 (90%)	84/84 (100%)			
generic	25.8k/59.8k (43%)	53.4k/59.8k (89%)	59.0k/59.8k (100%)			
	✗ link fails	✗ link fails	✓ running			
libpng	6/24 (25%)	6/24 (25%)	7/24 (29%)			
	25.4k/56.5k (45%)	25.4k/56.5k (45%)	3.1k/56.5k (5%)			
	✗ link fails	✗ link fails	✓ running			
zlib	5/17 (29%)	12/17 (71%)	17/17 (100%)			
	1.9k/10.8k (18%)	8.8k/10.8k (81%)	10.8k/10.8k (100%)			
	✓ running	✓ running	✓ running			
bzip2	3/9 (33%)	9/9 (100%)	9/9 (100%)			
	0.7k/6.9k (10%)	6.9k/6.9k (100%)	6.9k/6.9k (100%)			
	✗ link fails	✗ link fails	✓ running			
figlet	3/6 (50%)	6/6 (100%)	6/6 (100%)			
	0.9k/5.2k (18%)	5.2k/5.2k (100%)	5.2k/5.2k (100%)			
	✗ link fails	✓ running	✓ running			
lolcat	1/2 (50%)	1/2 (50%)	2/2 (100%)			
	0.1k/0.6k (14%)	0.1k/0.6k (14%)	0.6k/0.6k (100%)			
	✓ running	✓ running	✓ running			
smaz	2/2 (100%)	2/2 (100%)	2/2 (100%)			
	0.3k/0.3k (100%)	0.3k/0.3k (100%)	0.3k/0.3k (100%)			
	✓ running	✓ running	✓ running			
nyancat	0/1 (0%)	1/1 (100%)	1/1 (100%)			
	0.0k/0.9k (0%)	0.9k/0.9k (100%)	0.9k/0.9k (100%)			
	✗ build fails	✗ link fails	✓ running			
sl	0/1 (0%)	0/1 (0%)	1/1 (100%)			
	0.0k/0.3k (0%)	0.0k/0.3k (0%)	0.3k/0.3k (100%)			
	✗ build fails	✗ build fails	✓ running			
Overall	66/146 (45%)	113/146 (77%)	129/146 (88%)			
	55.1k/140.6k (39%)	101.3k/140.6k (72%)	87.2k/140.6k (62%)			
	C2Rust experimental	C2Rust	Oxidize			

	0	20	40	60	80	100
Constant literal	111/1132 (10%)	576/1132 (51%)	1108/1132 (98%)	1121/1132 (99%)		
	542/2850 (19%)	2011/2850 (71%)	0/2850 (0%)	2107/2850 (74%)		
Constant expression	22/68 (32%)	53/68 (78%)	65/68 (96%)	68/68 (100%)		
	27/359 (8%)	258/359 (72%)	0/359 (0%)	277/359 (77%)		
Non-constant expression	23/268 (9%)	119/268 (44%)	166/268 (62%)	235/268 (88%)		
	527/1746 (30%)	877/1746 (50%)	0/1746 (0%)	1034/1746 (59%)		
Function-like (expr)	0/101 (0%)	0/101 (0%)	0/101 (0%)	101/101 (100%)		
	0/1498 (0%)	0/1498 (0%)	0/1498 (0%)	543/1498 (36%)		
Function-like (stmt)	0/1 (0%)	1/1 (100%)	1/1 (100%)	1/1 (100%)		
	0/1911 (0%)	8/1911 (0%)	0/1911 (0%)	121/1911 (6%)		
Other	0/5 (0%)	0/5 (0%)	5/5 (100%)	4/5 (80%)		
	0/1438 (0%)	0/1438 (0%)	0/1438 (0%)	5/1438 (0%)		
Overall	156/1575 (10%)	749/1575 (48%)	1345/1575 (85%)	1530/1575 (97%)		
	109k/9802 (11%)	315k/9802 (32%)	0/9802 (0%)	4087/9802 (42%)		
	C2Rust experimental	C2Rust	rust-bindgen	Oxidize		

Qualified TUs (Oxidize n C2Rust):
bzip2 9/9, doomgeneric 76/84, figlet 6/6, libpng 1/24, lolcat 1/2, nyancat 1/1, sl 0/1, smaz 2/2, zlib 12/17

7. Limitations

- Macro definition clarity
- Large number of macro parameters
- Unsupported C features
- Statement-like or type-level macros

```
1 pub unsafe extern "C" fn skipws(mut fp: *mut ZFILE) -> () {
2     let mut c: i32 = (0 as i32);
3     while ({c = Zgetc!(fp, *mut ZFILE, i32, i32, *mut ZFILE, *mut u8, *mut ZFILE, i32, i32, u32) as i32}; (((isascii!(c, i32, i32, i32, i32, i32) != 0)
4             && (isspace!(c, *mut *const u16, *const u16, u32, u32) != 0)) as i32) != 0) {}
5     Zungetc!(c, fp, *mut ZFILE, *mut ZFILE);
6 }
```