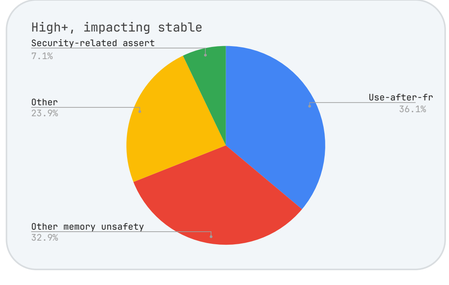
Oxidize: Macro-Aware C-to-Rust Translation

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

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

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

3. Solutions

Macro contraction — Parse everything as an expression

```c
#define LB {
#define BAR -1
int foo() LB
  return BAR BAR;
}
```

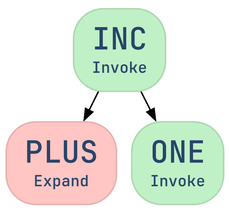
```c

int foo() {
  return -1 -1;
}
```

```rust
macro_rules! BAR {
() => { -1 }}
fn foo() -> i32 {
  BAR!() - 1
}
```

Invoke expand algorithm — Recursively expand necessary macros

```c
#define ONE 1
#define PLUS +
#define INC(a) a PLUS ONE
int foo() {
  return INC(1);
}
```




```c

#define ONE 1
#define INC(a) a + ONE
int foo() {
  return INC(1);
}
```

Generic helper traits — Redefine operators with C semantics

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

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

Passing in types — Types provided at invocation site

```c
#define ASSIGN(l, r) l = r
void baz() {
  int count;
  ASSIGN(count, (char)1);
}
```

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

Hygiene

- Check for identifier shadowing

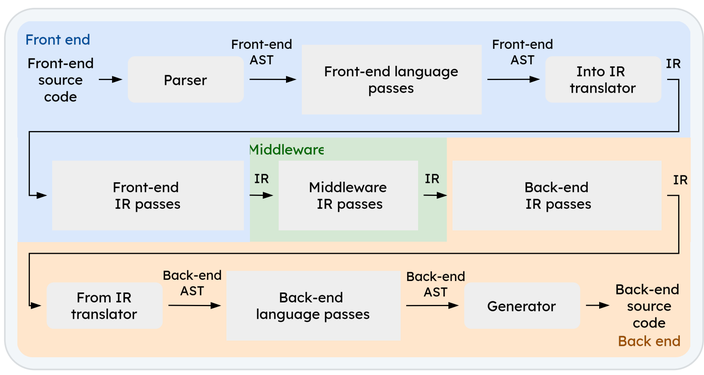
```h
int count = 0;
#define COUNT count
```

```c
#include <counter.h>
int foo() {
  int count = bar();
  return COUNT;
}
```

4. Oxidized example from §2

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

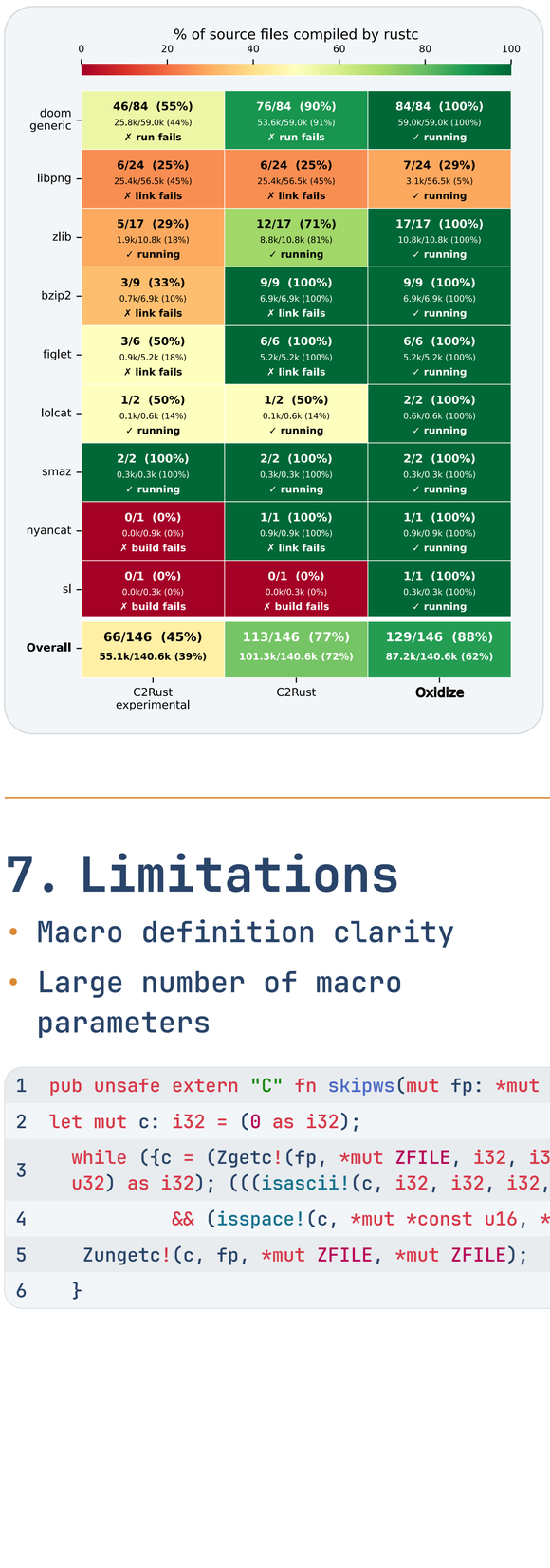
5. Oxidize architecture



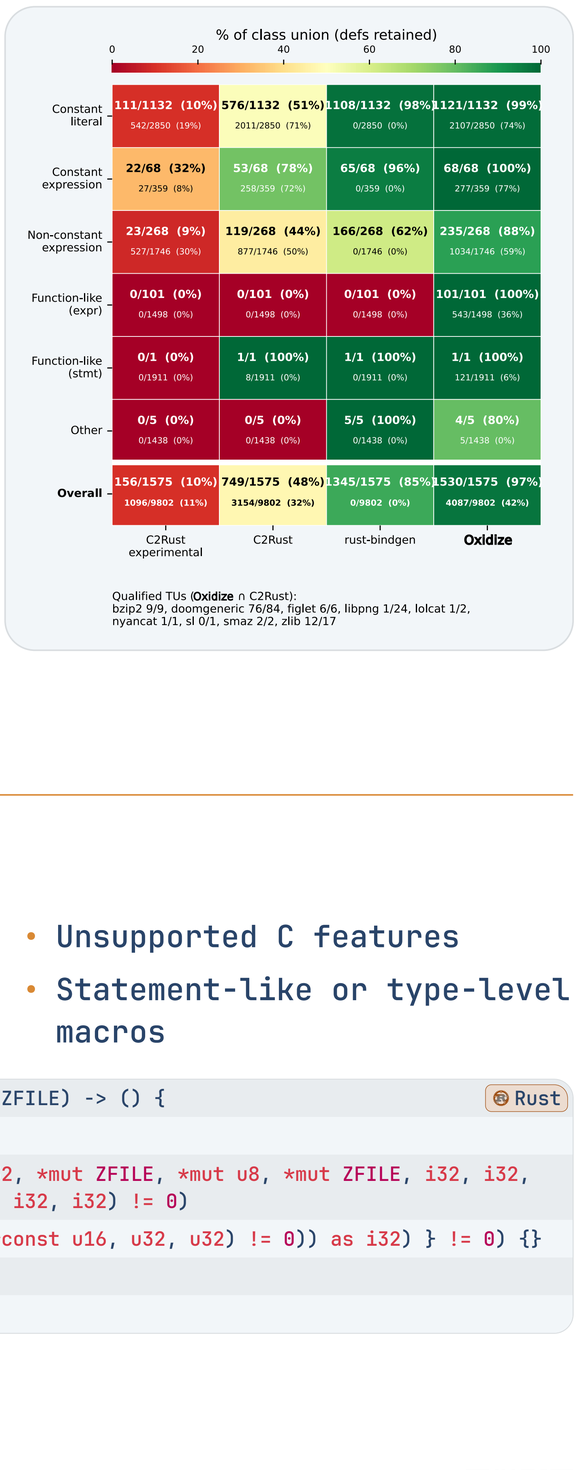
- Inspired by MLIR and the Nanopass framework
- 28 frontend passes and 13 backend passes

6. Evaluation

Codebase coverage



Macro retention



Qualified TUs (**Oxidize** ∩ 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

```rust
pub unsafe extern "C" fn skipws(mut fp: *mut ZFILE) -> () {
let mut c: i32 = (0 as i32);
  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)
          && (isspace!(c, *mut *const u16, *const u16, u32, u32) != 0)) as i32) } != 0) {}
   Zungetc!(c, fp, *mut ZFILE, *mut ZFILE);
  }
```

VRIJE UNIVERSITEIT BRUSSEL — SOFTWARE LANGUAGES LAB — Royal Military Academy Brussels

Our paper from LCTES'25 can be found here