Slides - 5/23

Justin Restivo

Table of contents

High Level Idea

Pitch

- Compile C to a unsafe subset of Rust (“RustLight”)

Pitch

- Compile C to a unsafe subset of Rust (“RustLight”)
- Run RustLight through the Rust compiler

Pitch

- Compile C to a unsafe subset of Rust (“RustLight”)
- Run RustLight through the Rust compiler
- RustLight operational semantics serve as a “Rust Spec”

Pitch

- Compile C to a unsafe subset of Rust (“RustLight”)
- Run RustLight through the Rust compiler
- RustLight operational semantics serve as a “Rust Spec”
- Improve on C2Rust

Translation Intuition (C2Rust)

Translation Intuition

- Run through examples from C2Rust

Types

- Types match using glibc types
- Some UB like addition overflow match Compcert C
- For pointers only use `*mut`

Calling convention

Calling convention: `extern "C"` or `extern "sysv64"`

Globals + Statics

C Lang

```c
static int global_counter = 0;
const int data = 0;

void increment_counter() {
    static int inner = 2;
    global_counter += data + inner;
}

```

Rust Lang

```rust
static mut global_counter: c_int = 0;
#[no_mangle]
pub static mut data: c_int = 0;

#[no_mangle]
pub unsafe extern "C" fn increment_counter()
{
    static mut inner: c_int = 2 as c_int;
    global_counter += data + inner;
}
```

Struct

C Lang

```c
struct LinkedList {
  int size;
  struct LinkedList* next;
  int data[];
};

```

Rust Lang

```rust
#[derive(Copy, Clone)]
#[repr(C)]
pub struct LinkedList {
    pub size: c_int,
    pub next: *mut LinkedList,
    pub data: [c_int; 0],
}
```

Union

C Lang

```c
union Data {
  int i;
  float f;
};
```

Rust Lang

```rust
#[derive(Copy, Clone)]
#[repr(C)]
pub union Data {
  pub i: c_int,
  pub f: c_float,
}
```

Loops

C Lang

```c
int main() {
    int i = 1;

    while (i <= 10) {
        i++;
    }

    return 0;
}

```

Rust Lang

```rust
unsafe fn main_0() -> c_int {
    let mut i: c_int = 1 as c_int;
    i = 0 as c_int;
    while i < 5 as c_int {
        i += 1;
        i;
        i += 1;
        i;
    }
    return 0 as c_int;
}

```

Switch

C Lang

```c
void copy_mod(char *to, const char *from, int count) {
    int n = (count + 2) / 3;

    switch (count % 3) {
        case 0:        *to++ = *from++;
        case 2:        *to++ = *from++;
        case 1:        *to++ = *from++;
    }
}

```

Rust Lang

```rust
#[no_mangle]
pub unsafe extern "C" fn copy_mod(
    mut to: *mut c_char,
    mut from: *const c_char,
    mut count: c_int,
) {
    let mut n: c_int = (count + 2 as c_int) / 3 as c_int;
    let mut current_block_2: u64;
    match count % 3 as c_int {
        0 => {
            let fresh0 = from;
            from = from.offset(1);
            let fresh1 = to;
            to = to.offset(1);
            *fresh1 = *fresh0;
            current_block_2 = 3977108684013665309;
        }
        2 => {
            current_block_2 = 3977108684013665309;
        }
        1 => {
            current_block_2 = 12446396083632624885;
        }
        _ => {
            current_block_2 = 715039052867723359;
```

Switch

```
        }
    }
    match current_block_2 {
        3977108684013665309 => {
            let fresh2 = from;
            from = from.offset(1);
            let fresh3 = to;
            to = to.offset(1);
            *fresh3 = *fresh2;
            current_block_2 = 12446396083632624885;
        }
        _ => {}
    }
    match current_block_2 {
        12446396083632624885 => {
            let fresh4 = from;
            from = from.offset(1);
            let fresh5 = to;
            to = to.offset(1);
            *fresh5 = *fresh4;
        }
        _ => {}
    };
}
```

Goto

C Lang

```c
int sample(int a) {
    int result = 0;

    if (a == 1) {
        goto answer_1;
    } else if (a == 2) {
        goto answer_2;
    }

answer_1:
    result = 1;
    goto end;

answer_2:
    result = 2;
    goto end;

end:
    return result;
}
```

Rust Lang

```rust
#[no_mangle]
pub unsafe extern "C" fn sample(mut a: c_int) -> c_int {
    let mut current_block: u64;
    let mut result: c_int = 0 as c_int;
    if a == 1 as c_int {
        current_block = 710105030588991595;
    } else if a == 2 as c_int {
        result = 2 as c_int;
        current_block = 2013428324500076459;
    } else {
        current_block = 710105030588991595;
    }
    match current_block {
        710105030588991595 => {
            result = 1 as c_int;
        }
        _ => {}
    }
    return result;
}

```

Duff's Device

C Lang

```c
void duffsDevice(char *to, const char *from, int count) {
    int n = (count + 2) / 3;

    switch (count % 3) {
        case 0: do {   *to++ = *from++;
        case 2:        *to++ = *from++;
        case 1:        *to++ = *from++;
                  } while (--n > 0);
    }
}
```

Rust Lang

```rust
#[no_mangle]
pub unsafe extern "C" fn duffsDevice( mut to: *mut c_char, mut from: *const c_char, mut count:
c_int) {
    let mut n: c_int = (count + 2 as c_int) / 3 as c_int;
    let mut current_block_2: u64;
    match count % 3 as c_int {
        0 => {
            current_block_2 = 12237857397564741460;
        }
        2 => {
            current_block_2 = 11244789108393354615;
        }
        1 => {
            current_block_2 = 6256153909998011048;
        }
        _ => {
            current_block_2 = 11875828834189669668;
        }
    }
    loop {
        match current_block_2 {
            11875828834189669668 => {
                return;
            }
            12237857397564741460 => {
                let fresh0 = from;
                from = from.offset(1);
                let fresh1 = to;
                to = to.offset(1);
```

Duff's Device

```
                *fresh1 = *fresh0;
                current_block_2 = 11244789108393354615;
            }
            11244789108393354615 => {
                let fresh2 = from;
                from = from.offset(1);
                let fresh3 = to;
                to = to.offset(1);
                *fresh3 = *fresh2;
                current_block_2 = 6256153909998011048;
            }
            _ => {
                let fresh4 = from;
                from = from.offset(1);
                let fresh5 = to;
                to = to.offset(1);
                *fresh5 = *fresh4;
                n -= 1;
                if n > 0 as c_int {
                    current_block_2 = 12237857397564741460;
                } else {
                    current_block_2 = 11875828834189669668;
                }
            }
        }
    };
}
```

Lifetimes & Existing Work

Lifetimes

- Claim: Use one pointer type `*mut`
- Need to prove RustLight lifetime semantics matches with Rust

Non-lexical Lifetimes

```
1    fn main() {
2        let mut scores = vec![1, 2, 3];
3        let score = &scores[0];
4        scores.push(4);
5    }
```

(Credit: SO)

Existing Work

Work	Supports NLL	Supports TPB	Is Source Level	Strictness wrt BC	Models Unsafe
RustBelt	Mostly	No	No	Not Strict Enough	Yes
Oxide	No	No	Yes	Too strict	No
K	No	No	Yes	Too strict	No
Stack Borrows	Yes	Yes	Yes	Too Strict	Yes
Tree Borrows	Yes	Yes	Yes	Slightly Too Strict	Yes

Two-phase borrow

```
// pub fn push(&mut self, value: T)
fn main() {
  let mut v = Vec::new();
  v.push(v.len());
  let r = &mut Vec::new();
  Vec::push(r, r.len());
}
```

(Credit: Rustc dev)

Tree Borrows

Each pointer is a state machine that is either:

- Reserved
- Active
- Disabled
- Frozen