COMP 737011 - Memory Safety and Programming Language Design

Lecture 10: Rust Toolchain

Hui Xu

xuh@fudan.edu.cn

Outline

1. Compiler: Rust Feature Implementation
2. Rust Toolchain

1. Compiler: Rust Feature Implementation

Rust Compiler

- Rustc: the official Rust compiler on top of LLVM
  - https://www.rust-lang.org/tools/install

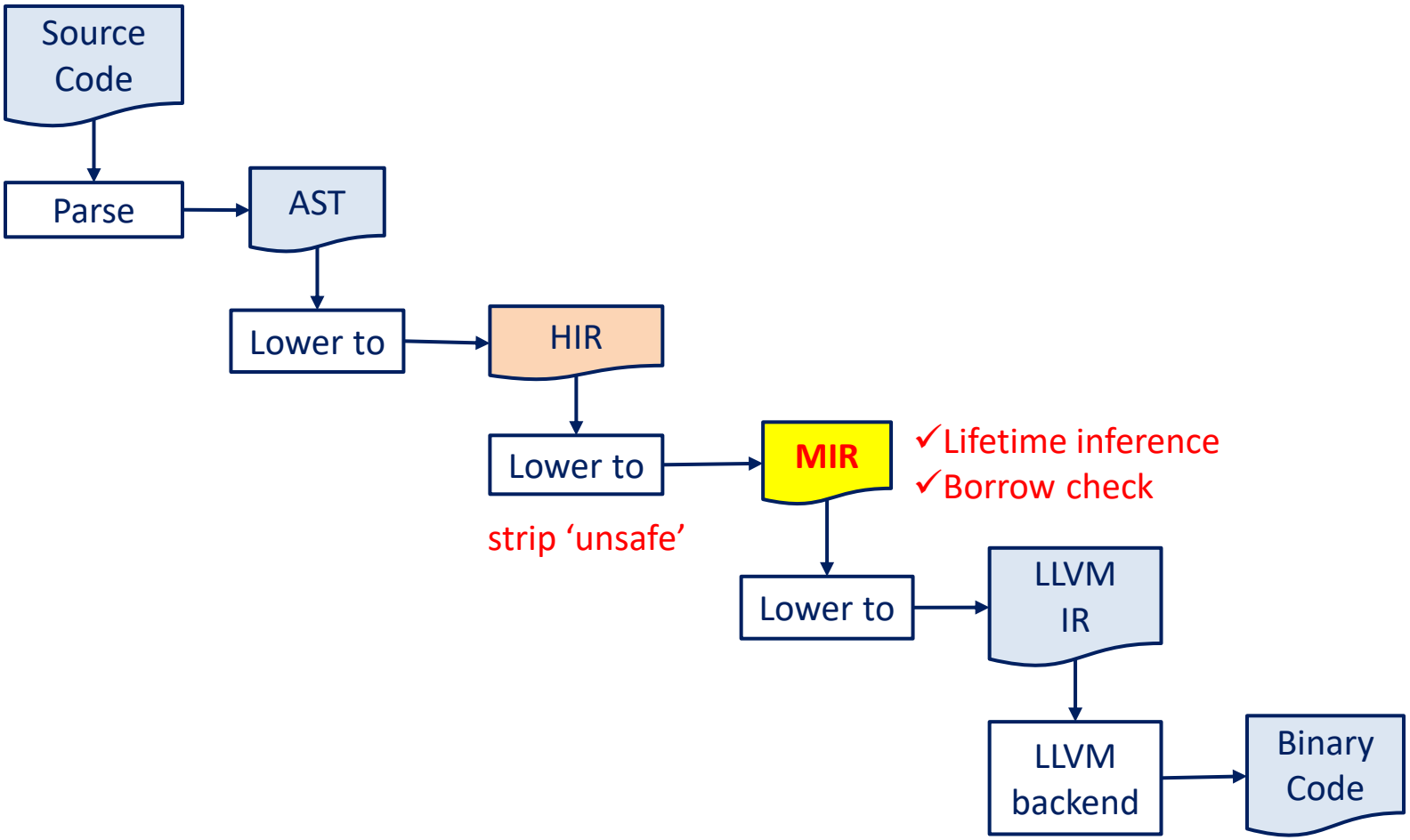
Using rustup (Recommended)

It looks like you're running macOS, Linux, or another Unix-like OS. To download Rustup and install Rust, run the following in your terminal, then follow the on-screen instructions. See "Other Installation Methods" if you are on Windows.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- Cranelift: an experimental backend for the Rust compiler written in Rust
  - https://github.com/rust-lang/rustc_codegen_cranelift
- gccrs: Rust compiler ontop of GCC
  - https://github.com/Rust-GCC/gccrs

Compilation Stages



https://rustc-dev-guide.rust-lang.org/compiler-src.html

HIR

- HIR is similar to AST (tree-based IR) but more succinct, e.g.,
  - Remove parenthesis
  - Convert “if let” to “match”
- Command to output HIR

```
#: rustc -Z help
...
#: rustc -Z unpretty=hir-tree toy.rs
Crate {
    item: CrateItem {
        module: Mod {
            inner: toy.rs:2:1: 5:2 (#0),
            item_ids: [
                ItemId {
                    id: HirId {
                        owner: DefId(0:1 ~ toy[317d]::{{misc}}[0]),
                        local_id: 0,
                    },
                },
...
```

MIR

- MIR is linear IR

```
fn main() {
    let alice = Box::new(1);
    let bob = &alice;
}
```

```
#: rustc -Z dump-mir=all toy.rs
```

```
fn main() -> () {
    let mut _0: ();
    let _1: std::boxed::Box<i32>;
    scope 1 {
        debug alice => _1;
        let _2: &std::boxed::Box<i32>;
        scope 2 {
            debug bob => _2;
        }
    }
    bb0: {
        StorageLive(_1);
        _1 = const std::boxed::Box::<i32>
             ::new(const 1_i32)
             -> [return: bb2, unwind: bb1];
    }
    bb1 (cleanup): {
        resume;
    }
    bb2: {
        FakeRead(ForLet, _1);
        StorageLive(_2);
        _2 = &_1;
        FakeRead(ForLet, _2);
        _0 = const ();
        StorageDead(_2);
        drop(_1) -> [return: bb3, unwind: bb1];
    }
    bb3: {
        StorageDead(_1);
        return;
    }
}
```

return value

assignment

borrow

https://rust-lang.github.io/rfcs/1211-mir.html

Unsafe Code Handling

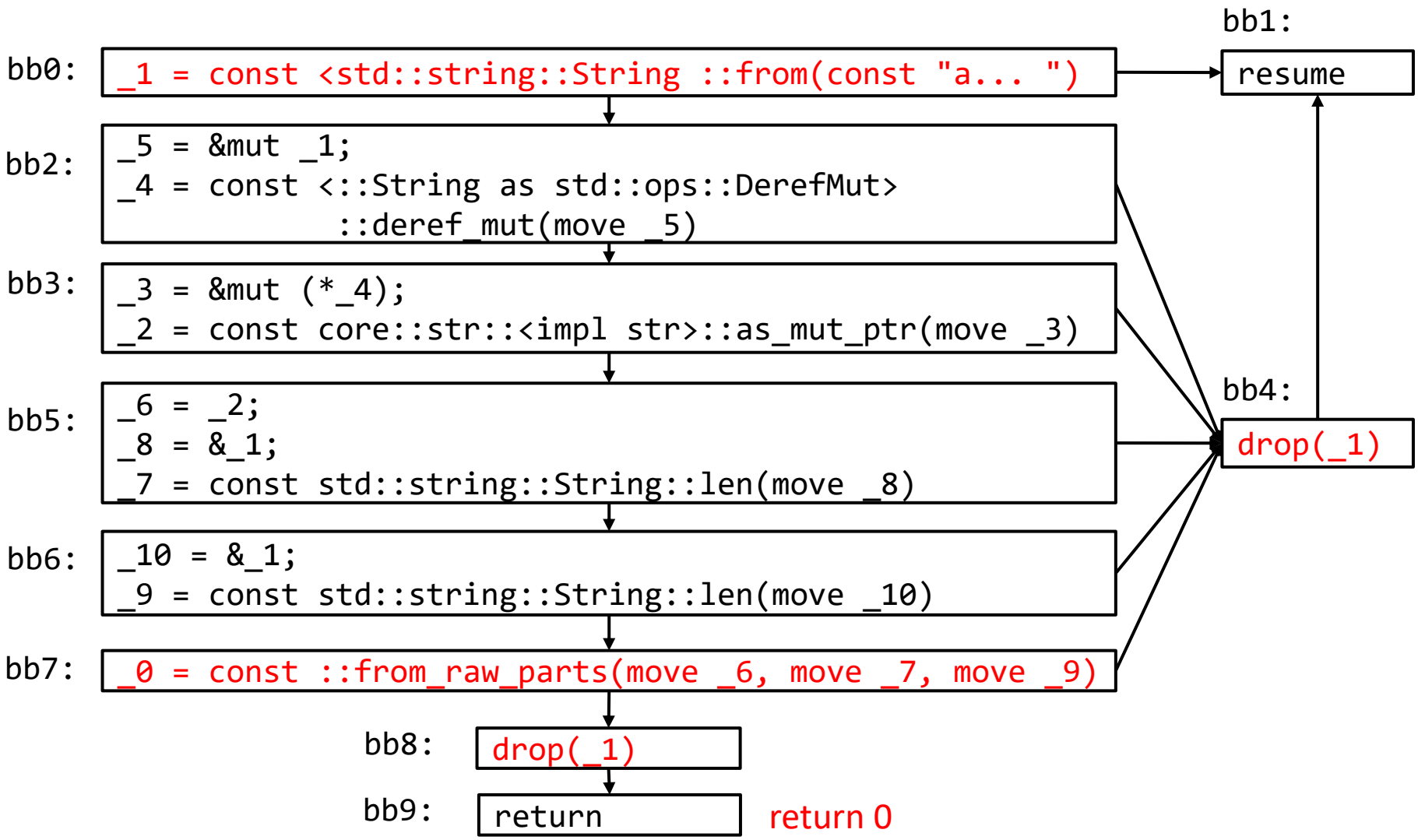
- Unsafe marker is stripped away in MIR.
- Raw pointers may introduce shared mutable aliases.

PoC of CVE-2019-16140

```
fn genvec()->Vec<u8>{
    let mut s = String::from("a tmp string");
    let ptr = s.as_mut_ptr();
    unsafe{
        let v = Vec::from_raw_parts(ptr,s.len(),s.len());
        v
    }
}
fn main(){
    let v = genvec(); //v is dangling
    println!("{:?}",v); //illegal memory access
}
```

```
#:./uaf
[104, 16, 195, 158, 247, 85, 0, 0, 0, 0, 0, 0]
Segmentation fault (core dumped)
#: rustc -V
rustc 1.44.1 (c7087fe00 2020-06-17)
```

MIR Analysis



Bug Fix

```
fn genvec2()->Vec<u8>{
    let mut s = String::from("a tmp string");
    let ptr = s.as_mut_ptr();
    unsafe{
        let v = Vec::from_raw_parts(ptr,s.len(),s.len());
        std::mem::forget(s);
        v
    }
}
```

```
_6 = const std::vec::Vec::<u8>::from_raw_parts(move _7, move _8, move _10)
```

↓

```
_13 = move _1;
_12 = const std::mem::forget::<std::string::String>(move _13)
```

↓

```
_0 = move _6;
```

↓

```
return;
```

Lifetime Inference

- Lifetimes are not based on lexical scopes or blocks.
- Problem: how to infer the minimum lifetime of each reference?
- Soundness requirement: The lifetime of each reference should not exceed its referent value.

```
fn main() { //scope starts
    let mut alice = Box::new(1);
    let bob = &alice;
    *alice = 2;
} //scope ends
```

bob is alive only in this statement

https://rust-lang.github.io/rfcs/2094-nll.html

Constraint-based Lifetime Inference

- Lifetime constraint extraction:
  - Liveness: L @ P: lifetime L is alive at the point P.
  - Subtyping: L1: L2 @ P: lifetime L1 outlives lifetime L2 at point P
- Constraint solving.
  - Reject the program if there is no solution.

```
let mut alice = Box::new(1);
let bob = &alice;
*alice = 2;
```

Constraints:

```
'alice @ BB0/1
'bob @ BB0/2
'alice: bob @ BB0/2
'alice @ BB0/3
```

⇩

```
'alice ⊇ {BB0/1, BB0/2, BB0/3}
'bob ⊇ {BB0/2}
```

Another Example with Branches

```
let mut a: i32 = 1;
let mut b: i32 = 2;
let mut p = &a;
// program point 1
if condition {
    // program point 2
    print(*p);
    // program point 3
    p = &b;
    // program point 4
}
// program point 5
print(*p);
// program point 6
```

p (&a) is alive.

p (&a) is alive.

p (&a) is dead

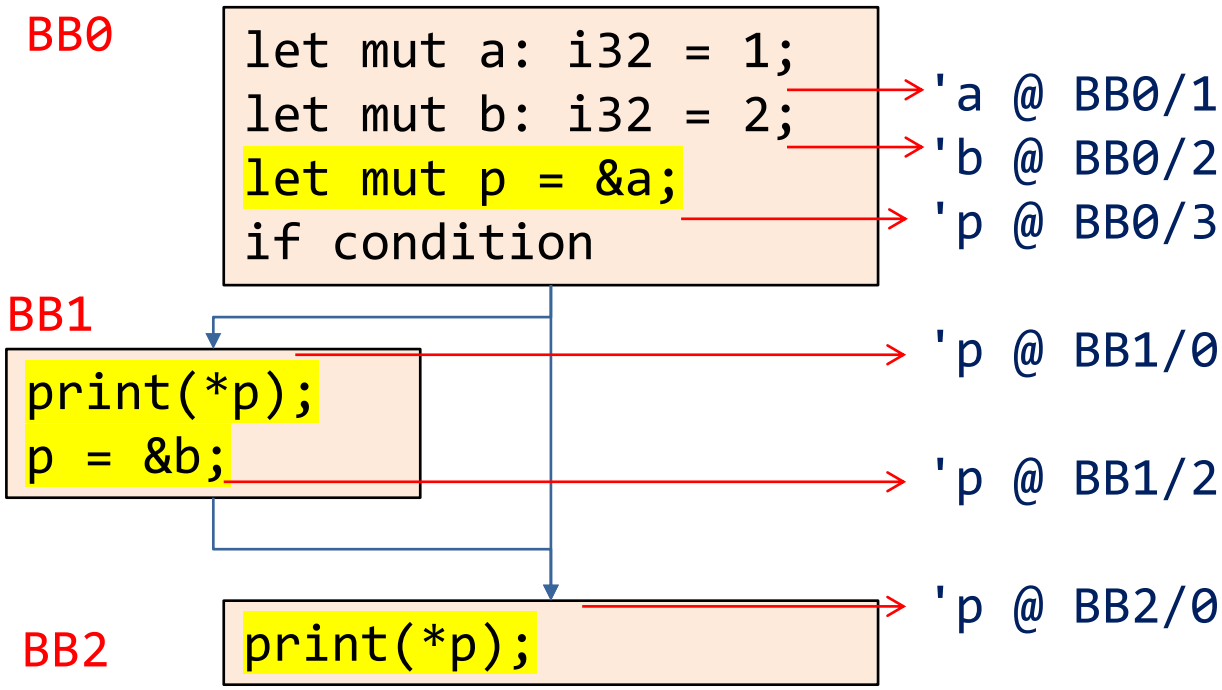
p (&b) is alive

p (&a or &b) is alive

p (&a or &b) is dead

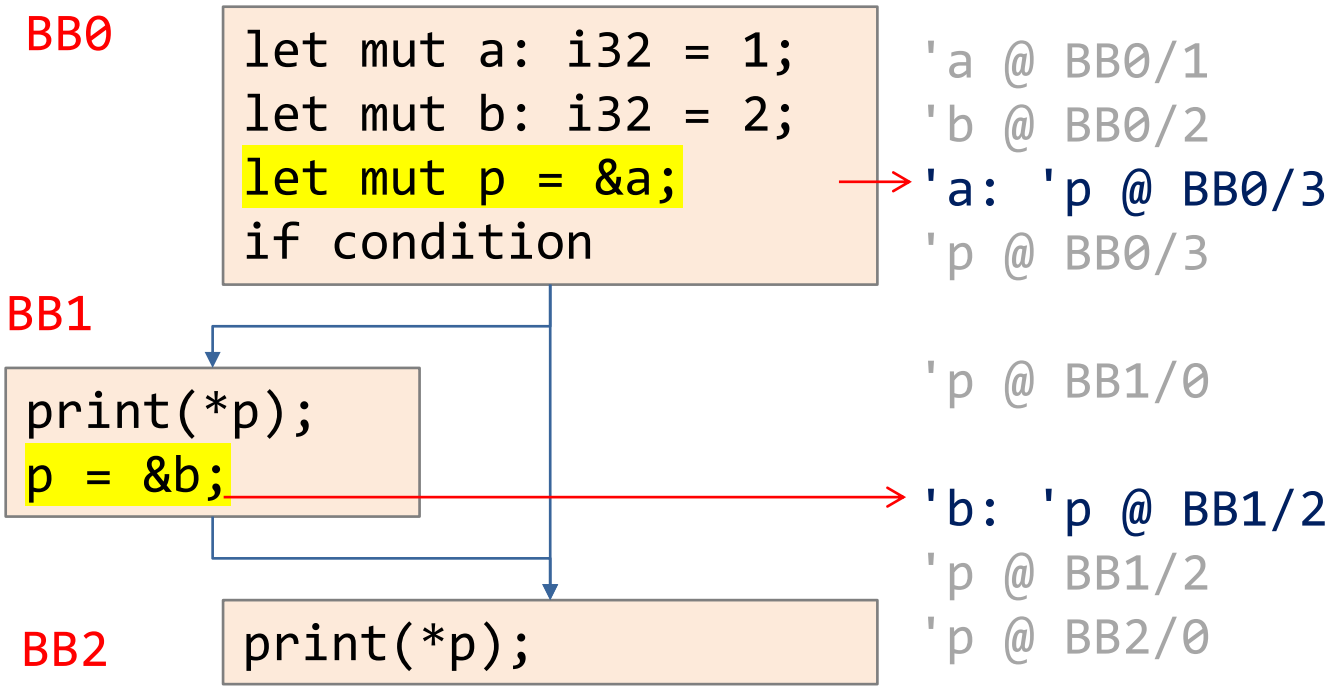
Constraint Representation: Liveness

- (L: {P}) @ P: lifetime L is alive at the point P.



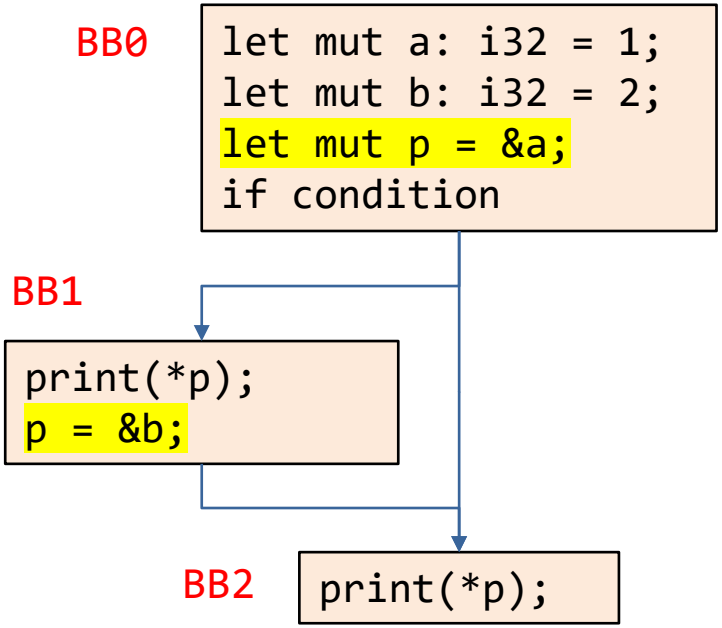
Constraint Representation: Subtyping

- (L1: L2) @ P: lifetime L1 outlives lifetime L2 at point P



Solving Constraints via Fixed-Point Iteration

- Init each lifetime variable with an empty set.
- Iterate over the constraints via depth-first search.
- Stop until all constraints are satisfied.




```
'a @ BB0/1
'b @ BB0/2
'a: 'p @ BB0/3
'p @ BB0/3

'p @ BB1/0

'b: 'p @ BB1/2
'p @ BB1/2
'p @ BB2/0
```

```
'p = {BB0/3-4, BB1/0-2, BB2/0}
'a = {BB0/1-4, BB1/0-2, BB2/0}
'b = {BB0/2-4, BB1/0-2, BB2/0}
```

More Rules

- We should define the constraint extraction rule for each particular type of statement.
- Reborrow constraint is complicated…

```
let mut a: i32 = 1;    'a @ BB0/1
let mut b = &a;        'b @ BB0/2 'a: 'b @ BB0/2
let c = &*b;           'c @ BB0/3 'b: 'c @ BB0/3
                       => also implies ('a: 'c)
```

```
let mut a: i32 = 22;     'a @ BB0/1
let mut b = &a;          'b @ BB0/2 'a: 'b @ BB0/2
let c = &mut b;          'c @ BB0/3 'b: 'c @ BB0/3
let d = &**c;            'd @ BB0/4 'c: 'd @ BB0/4
                         => also implies ('b: 'd), ('a: 'd)
```

False Rejections?

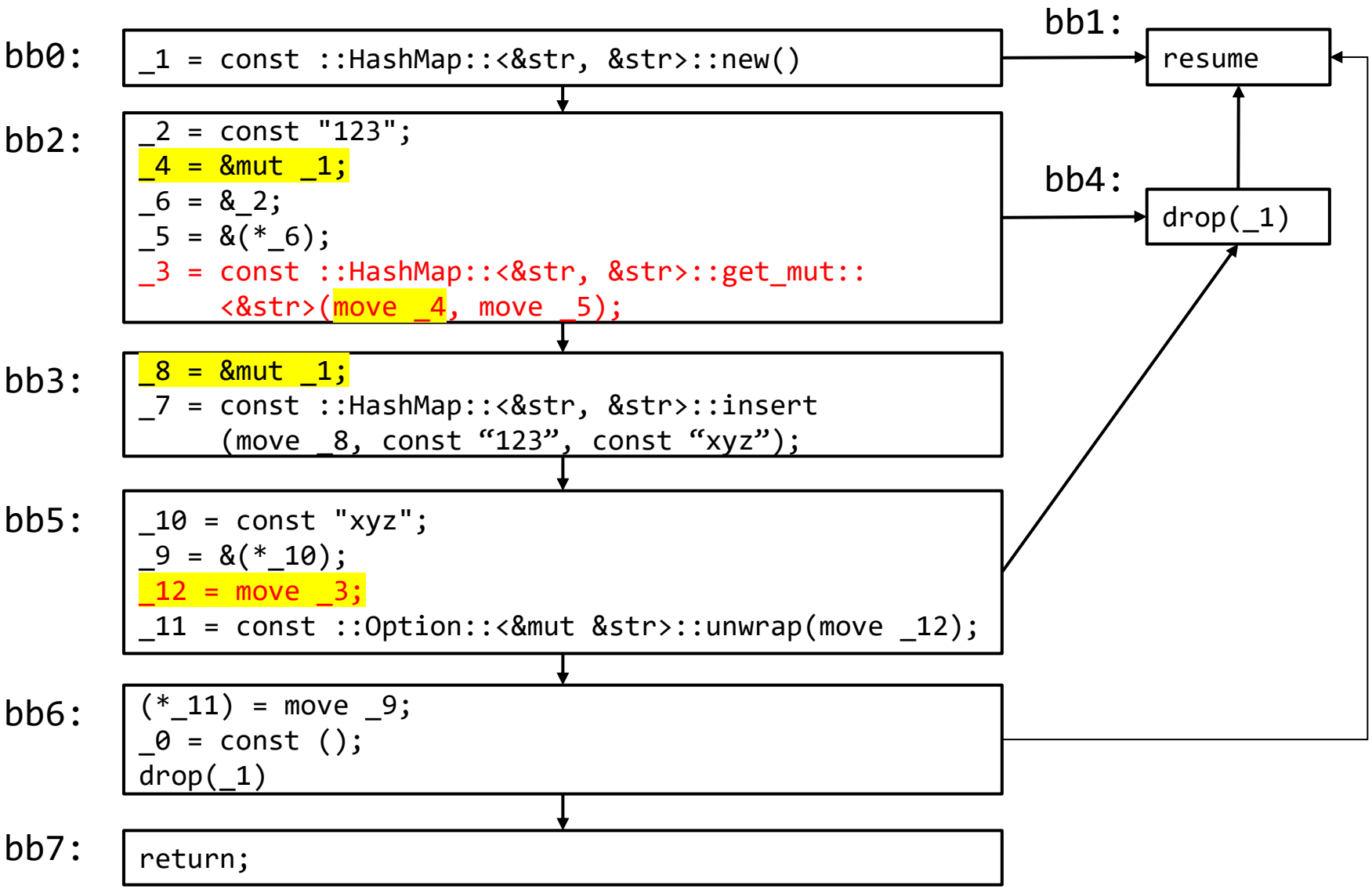
```rust
let mut map = HashMap::new();
let key = "123";
let v =  map.get_mut(&key);
map.insert("234","xyz");
*(v.unwrap()) = "xyz";
```

```rust
pub fn get_mut<Q>(&mut self, k: &Q) -> Option<&mut V>
where
    K: Borrow<Q>,
    Q: Hash + Eq + ?Sized,
```

Before lifetime elision: &'a self, &'b Q, &'a V

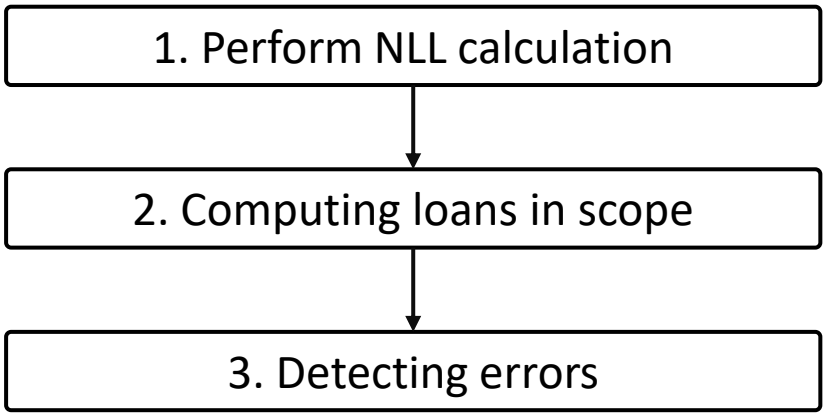
```
error[E0499]: cannot borrow `map` as mutable more than once at a time
 --> nll_case.rs:7:5
6 |     let v =  map.get_mut(&key);
  |              ---------------- first mutable borrow occurs here
7 |     map.insert("123","xyz");
  |     ^^^^^^^^^^^^^^^^^^^^^^^ second mutable borrow occurs here
8 |     *(v.unwrap()) = "xyz";
  |       - first borrow later used here
```

Analysis Based on MIR



Borrow Check

- Check based on the MIR
- Older implementation was on the HIR



Computing Loans in Scope: Transfer Function

- Any loans whose region does not include the point are killed.
- For a borrow statement, the corresponding loan is generated.
- For statement lv = <rvalue>, killed the loan of which lv is a prefix

https://rustc-dev-guide.rust-lang.org/borrow_check.html

Rules to Detect Errors

- All variables should be initialized before they are used.
- You can't move the same value twice.
- You can't move a value while it is borrowed.
- You can't read/write a place while it is mutably borrowed.
- You can't mutate a place while it is immutably borrowed.

Sample Analysis Approach

bb0:

```
_1 = const ::HashMap::<&str, &str>::new()
```

O={1}

bb2:

```
_2 = const "123";
_4 = &mut _1;
_6 = &_2;
_5 = &(*_6);
_3 = const ::HashMap::<&str, &str>::get_mut::
     <&str>(move _4, move _5);
```

O={1}, MB={4}

O={1}, MB={3}

bb3:

```
_8 = &mut _1;
_7 = const ::HashMap::<&str, &str>::insert
     (move _8, const “123”, const “xyz”);
```

O={1}, MB={3,8}

O={1}, MB={3,7}

Conflict!!!

bb5:

```
_10 = const "xyz";
_9 = &(*_10);
_12 = move _3;
_11 = const ::Option::<&mut &str>::unwrap(move _12);
```

O={1}, MB={3}

O={1}, MB={12}

O={1}, MB={11}

bb6:

```
(*_11) = move _9;
_0 = const ();
drop(_1)
```

O={1}

bb7:

```
return;
```

2. Rust Toolchain

Rust Toolchain

- **rustc – The Rust compiler**
  - Compiles .rs source files into executable binaries.
- **rustup – The toolchain installer & version manager**
  - Manages Rust versions, channels (stable/beta/nightly), and related tools.
- **cargo – The package manager & build tool**
  - Handles building, running, testing, and dependency management.
- **crates.io – The official Rust package registry**
  - Source of reusable libraries ("crates") for your project.
- **docs.rs – Hosted documentation for published crates**

Rustup

```
#: rustup show
Default host: x86_64-unknown-linux-gnu
rustup home:  /home/aisr/.rustup

installed toolchains
--------------------

stable-x86_64-unknown-linux-gnu
nightly-2023-10-05-x86_64-unknown-linux-gnu
nightly-2024-06-30-x86_64-unknown-linux-gnu
nightly-2024-09-05-x86_64-unknown-linux-gnu
nightly-2024-10-12-x86_64-unknown-linux-gnu (default)

#: rustup default stable-x86_64-unknown-linux-gnu
info: using existing install for 'stable-x86_64-unknown-linux-gnu'
info: default toolchain set to 'stable-x86_64-unknown-linux-gnu'
```

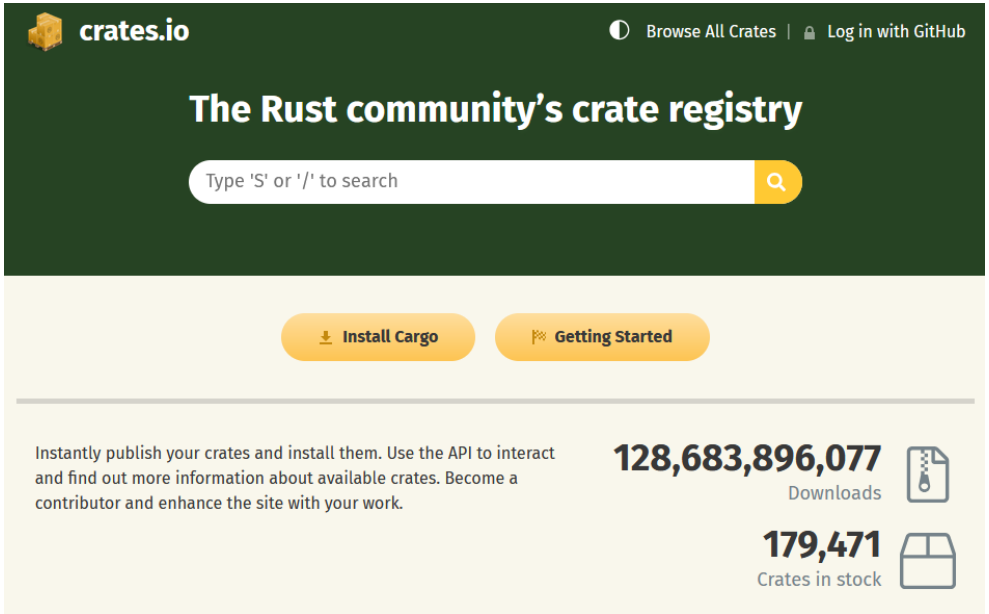
Cargo

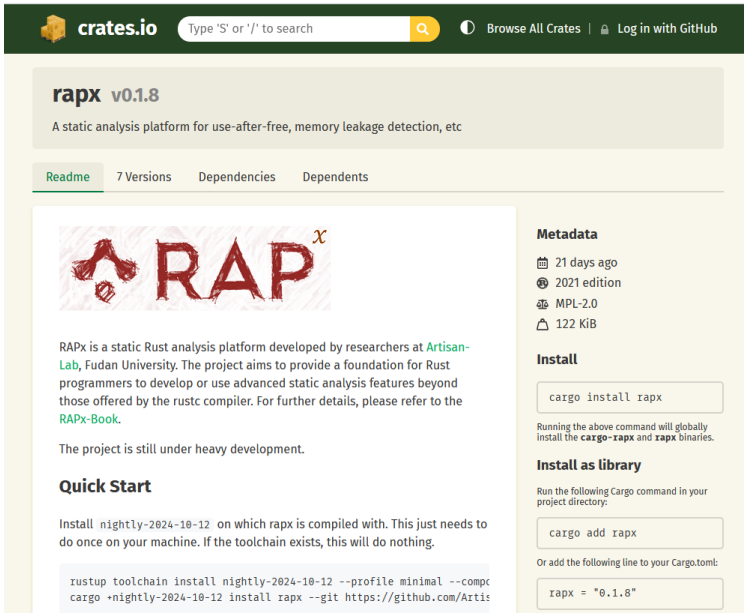
- Configuration files: Cargo.toml
  - Workspace, targets, dependencies, etc.
- Features:
  - Build your Rust code: cargo build
  - Run your programs: cargo run
  - Run tests: cargo test
  - More, including some third-party features published to crates.io

```
#: cargo new foo
    Creating binary (application) `foo` package
note: see more `Cargo.toml` keys and their definitions at
https://doc.rust-lang.org/cargo/reference/manifest.html
#: cd foo build
#: cargo build
   Compiling foo v0.1.0 (/home/aisr/foo)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.76s
```

Crates.io

- The official public registry for Rust packages, known as *crates*.
- It’s where Rust developers publish, discover, and reuse libraries.






```
#: cargo install rapx
...
#: cargo rapx -help
13:29:41|RAP|INFO|:
Usage:
    cargo rapx [rapx options] -- [cargo check options]
```

In-Class Practice

- Write a Rust program with use-after-free bugs.
- Emit the MIR code
- Analyze the MIR and discuss how to detect the bug.