Klint

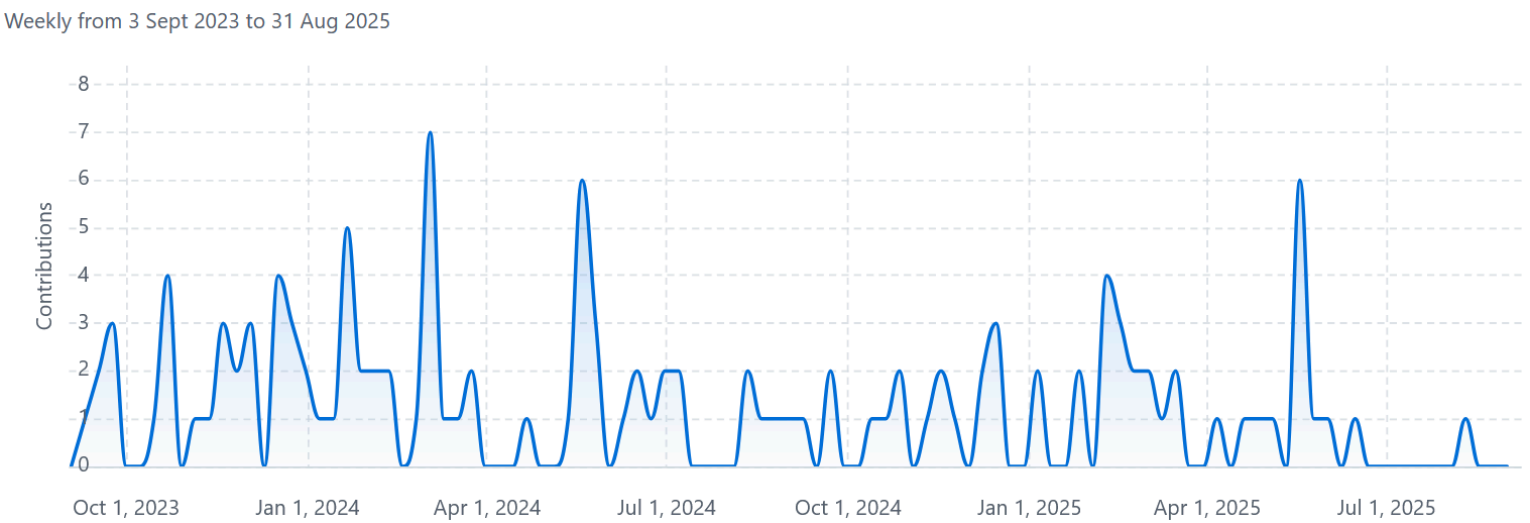
Gary Guo

Kangrejos 2025

Preempt Count Verifier Updates

Updates

- Mostly just maintenance
- Daily CI to catch upstream Rust changes
- Tagged release for each stable Rust version
- Some diagnostics improvement



Annotation shorthands

Previously, `klint` uses a range syntax to declare possible preemption count ranges when calling a function. Now a more friendly shorthand syntax is supported for most common cases:

```
#[klint::preempt_count(expect = ..)]   →  #[klint::atomic_context]
#[klint::preempt_count(expect = 1..)]  →  #[klint::atomic_context_only]
#[klint::preempt_count(expect = 0)]    →  #[klint::process_context]
```

Additionally `#[klint::any_context]` is reserved for functions that can be called in raw atomic context too (checks are not yet implemented for raw atomic contexts).

New supported pattern: if-let

This is previously unsupported, but is now handled correctly by klint:

```
#[klint::atomic_context]
fn foo(x: Option<SleepOnDrop>) -> Option<SleepOnDrop> {
    if let Some(x) = x {
        return Some(x);
    }
    None
}
```

This is a result of Rust MIR generation improvement, nothing is done specially at klint side.

Still unsupported pattern

This pattern is unfortunately still unsupported:

```
#[klint::atomic_context]
fn foo(
  x: Result<SleepOnDrop, SleepOnDrop>
) -> Result<SleepOnDrop, SleepOnDrop> {
    if let Ok(x) = x {
        return Ok(x);
    }
    x
}
```

Cause: MIR code generation

After lowering, Rust will generate code like this:

```
#[klint::atomic_context]
fn foo(
  x: Result<SleepOnDrop, SleepOnDrop>
) -> Result<SleepOnDrop, SleepOnDrop> {
    if x.is_ok() {
      ret = Ok((x as Ok).0); // This just moves a single field
      // This is a generated drop to drop the rest of fields (happens to be always no-op)
      match x {
        Ok(_) => (),
        _ => drop(x), // This unfortunately suppress any optimization on `x`.
      }
      return;
    }
    ret = x;
    return;
}
```

Cause: MIR code generation

Why would such MIR code being generated?

- Drop elaboration generates redundant drops
- `drop` takes mutable reference
- Rust MIR optimization are very pessimistic when there are mutable reference
- Notably, this inhibits variant tracking, causing the code to be unoptimized

This is not an issue for actual codegen as LLVM can optimize this away.

This however causes significant challenge to klint, as it now consider the dropping codepath to be possible.

Possible solution

Better variant tracking:

- Instead of disabling variant tracking when a place is mutably borrowed, only disable it *after* it is mutably borrowed.

Memory-model-aware optimization:

- Make optimization assumptions based on stacked borrow or tree borrow; klint can perform analysis on its own clone of the MIR, so such assumptions would not affect codegen.

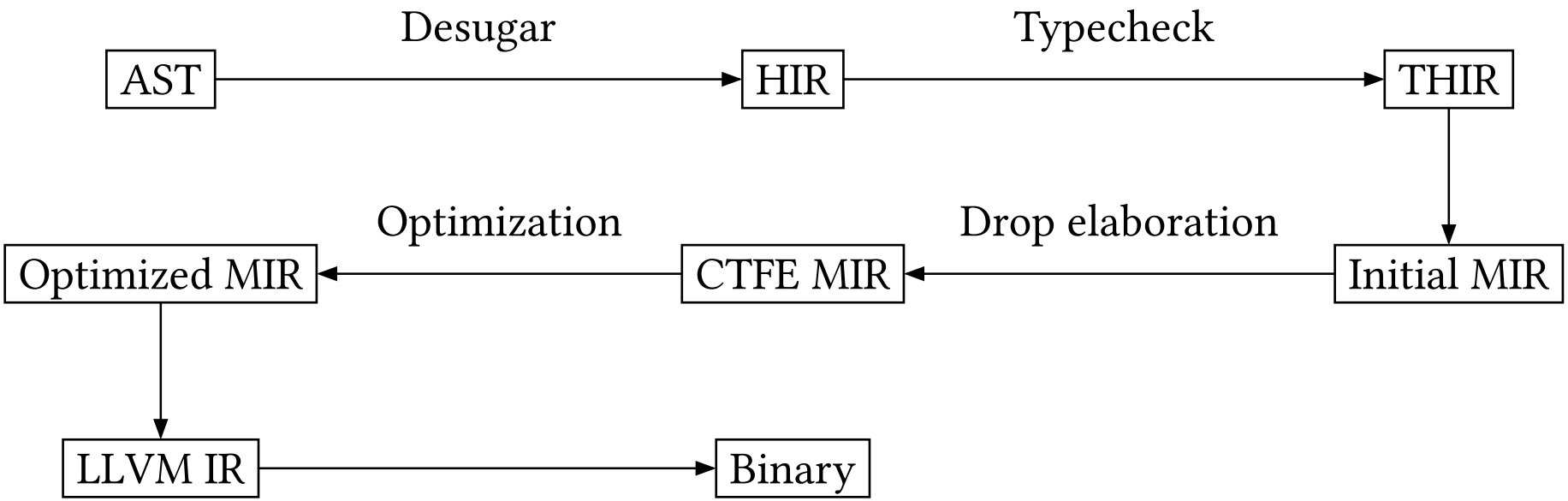
Still unsupported pattern

```
fn foo(lock: &SpinLock<()>) {
    loop {
        let guard = lock.lock();
        if random() {
            drop(guard);
            if random() {
                return;
            }
            continue;
        }
        return;
    }
}
```

Usually Rust insert drop flags if not all control flow path will drop a variable.

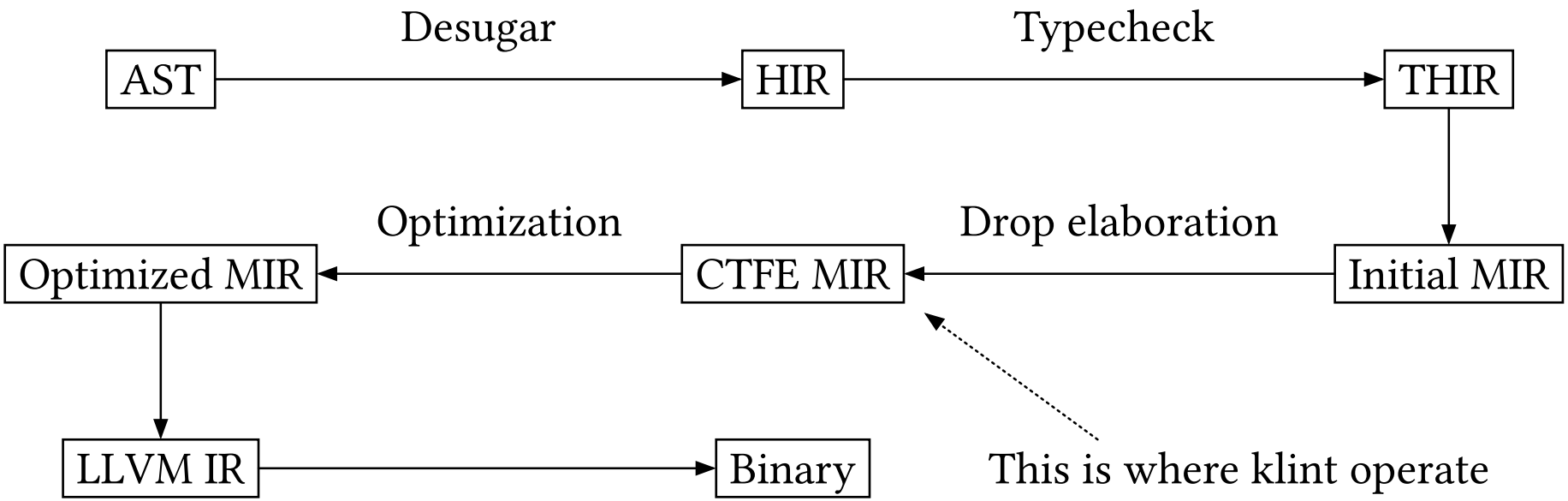
Cause: how the analysis is performed

Rust code goes through the following intermediate representations



Cause: how the analysis is performed

Rust code goes through the following intermediate representations



Drop-elaborated MIR

```
  fn foo(lock: &SpinLock<()>) {
      loop {
          let guard = lock.lock();
          let guard_needs_drop = true;
          if random() {
              guard_needs_drop = false;
              drop(guard);
              if random() {
                  goto ret;
              }
              continue;
          }
          goto ret;
      }
ret: // <-- at this point, we might have executed the dtor or might not
      if guard_needs_drop {
        drop(guard);
      }
  }
```

Possible solution: basic block specialization

```
fn foo(lock: &SpinLock<()>) {
    loop {
        let guard = lock.lock();
        let guard_needs_drop = true;
        if random() {
            guard_needs_drop = false;
            drop(guard);
            if random() {
                goto ret.0;
            }
            continue;
        }
        goto ret.1;
    }
ret.1:
    drop(guard);
ret.0:
    return;
}
```

Verifying the Rust Binder

Bug found

https://android-review.googlesource.com/c/kernel/common/+/3723545

```
// This can be called from atomic context
pub(crate) fn reservation_commit(&mut self, offset: usize, data: Option<T>) -> Result {
    // This could use a binary search, but linear scans are usually faster for small arrays.
    let range = self
        .ranges
        .iter_mut()
        .find(|range| range.offset == offset)
        .ok_or(ENOENT)?;
    let DescriptorState::Reserved(reservation) = &range.state else {
        return Err(ENOENT);
    };
    range.state = DescriptorState::Allocated(reservation.clone().allocate(data));
    Ok(())
}
```

Bug found

https://android-review.googlesource.com/c/kernel/common/+/3723545

```
// This can be called from atomic context
pub(crate) fn reservation_commit(&mut self, offset: usize, data: Option<T>) -> Result {
    // This could use a binary search, but linear scans are usually faster for small arrays.
    let range = self
        .ranges
        .iter_mut()
        .find(|range| range.offset == offset)
        .ok_or(ENOENT)?;
    let DescriptorState::Reserved(reservation) = &range.state else {
        return Err(ENOENT);
    };
    range.state = DescriptorState::Allocated(reservation.clone().allocate(data));
    Ok(())
}
```

Implicit drops that can sleep are tricky!

Not-a-bug found

https://android-review.googlesource.com/c/kernel/common/+/3736026

```
pub(crate) fn find_from(&self, thread: &Thread) -> Option<DArc<Transaction>>;

// This can be called from atomic context
let mut t_opt = inner.current_transaction.clone();
while let Some(t) = t_opt {
    if Arc::ptr_eq(&t.from, self) {
        t.debug_print_inner(m, "    outgoing transaction ");
        t_opt = t.from_parent.clone();
    } else if Arc::ptr_eq(&t.to, &self.process) {
        t.debug_print_inner(m, "    incoming transaction ");
        t_opt = t.find_from(self);
    } else {
        t.debug_print_inner(m, "    bad transaction ");
        t_opt = None;
    }
}
```

Not-a-bug found

https://android-review.googlesource.com/c/kernel/common/+/3736026

This code is actually okay, although klint is struggling to understand that you can hold an `Arc`, clone it, and drop it, and it would not actually drop the instance.

- Changing the signature of `find_from` to

  ```
  pub(crate) fn find_from(&self, thread: &Thread) -> Option<&DArc<Transaction>>;
  ```

  however eliminates the possibility of sleeping and generate better code as there's no need to increase and decrease the reference count!

Bug or not a bug?

- klint complains that Binder calls `sched_setscheduler_nocheck` inside atomic context.
- `sched_setscheduler_nocheck` can sleep if policy is `SCHED_DEADLINE`
  - `cpuset_lock` is called for this policy, and this takes the `cpuset_mutex`.
- Binder does not actually support `SCHED_DEADLINE`.
  - It checks that `SCHED_FIFO`/`SCHED_RR`/`SCHED_NORMAL`/`SCHED_BATCH` is used.

This is a case where whether a function can sleep or not depends on the value.

Is this a bug? We can:

- Specify the function to be called only inside process context and say Binder has a bug.
- Specify the function to be callable from atomic context and say it is wrongly implemented.
- Say this is intended behaviour and is just a klint deficiency.

New Feature ✨

Unhelpful `build_error` error messages

We have a `build_assert!` to perform some checks at build time without introducing possible runtime `BUG`s.

- This is checked by the linker for builtin modules

  A somewhat useful error message

```
ld.lld: error: undefined symbol: rust_build_error
>>> referenced by build_assert.rs:78 (rust/kernel/build_assert.rs:78)
>>>               samples/rust/rust_driver_pci.o:(<rust_driver_pci::SampleDriver as
kernel::pci::Driver>::probe) in archive vmlinux.a
```

- This is checked by `modpost` for loadable modules.

  A useless error message

```
ERROR: modpost: "rust_build_error" [samples/rust/rust_driver_pci.ko] undefined!
```

Post-codegen analysis ✨

- New capability of klint
- klint is now capable of checking the generated object files

Post-codegen `build_error` checks

```
error: this `build_error` reference is not optimized away
   --> /home/gary/Projects/linux/rust/kernel/io.rs:221:9
    |
221 |         build_assert!(Self::offset_valid::<U>(offset, SIZE));
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = note: inside instance `kernel::io::Io::<16>::io_addr_assert::<u32>`
note: which is called from here
   --> /home/gary/Projects/linux/rust/kernel/io.rs:228:5
    |
228 |     define_read!(read32, try_read32, readl -> u32);
    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    = note: inside instance `kernel::io::Io::<16>::read32`
note: which is called from here
   --> /home/gary/Projects/linux/samples/rust/rust_driver_pci.rs:48:9
    |
 48 |         bar.read32(Regs::END); // should trigger build error
    |         ^^^^^^^^^^^^^^^^^^^^
note: which is called from here
   --> /home/gary/Projects/linux/samples/rust/rust_driver_pci.rs:94:13
    |
 94 |             Self::testdev(info, bar)?
    |             ^^^^^^^^^^^^^^^^^^^^^^^^
note: reference contained in `fn <SampleDriver as kernel::pci::Driver>::probe`
   --> /home/gary/Projects/linux/samples/rust/rust_driver_pci.rs:70:5
    |
 70 |     fn probe(pdev: &pci::Device<Core>, info: &Self::IdInfo) -> Result<Pin<KBox<Self>>> {
    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Post-codegen `build_error` checks

This is performed immediately after generating object files.

- Rust compiler contexts are still kept around[1], and it has much richer information of the source.
- Object files generated are loaded back and relocations analyzed to spot `build_error` presence.
- DWARF debug information is used to recover inlining information and source locations; this is then cross-checked with MIR to recover accurate span information.

No need to wait until linking. Also provides a bit more useful line numbers than the linker.

[1] In normal Rust compilation flow, the `TyCtxt` is already destroyed at this point. I hacked the flow by replacing the codegen backend with a wrapper that changes the execution flow slightly.

Future Possibilities

Future venues of improvements

- Improving atomic context checks to cover more patterns
- Add support for `try_access` methods
- Additional post-codegen lints
  - Imagine `objtool`, but inside compiler and have access to internals.
- Other additional function colouring / effect systems
- Upstream features to rustc and remove local hacks
  - E.g. ability to keep klint metadata in rmeta file