Interviewer
What have been your motivations for learning Rust and choosing to use it?

Participant
My motivation for learning Rust was the better or simpler tooling compared to Ada at the time.

Interviewer
Could you describe a bit more about the differences in tooling and what you found great in Rust compared to what you found less great in Ada?

Participant
So I mean, the very first starting point was cargo. Ada has a build system. It's not make file based, but it is very hard to actually split things into multiple libraries and cargo that was just out of the box trivial to do and yeah, it even had uploading your libraries to share them with other people and everything. By now Ada has caught up, but back then that was not even on the radar. So there was, yeah.

4:1 So I mean, t...

Rust Has Great Tooling

Interviewer
Gotcha. So are there any aspects aside from the package management systems in terms of the language like the memory safety guarantees with Rust that were a benefit or was it mostly just the improved?

Participant
Other tooling too, but like from the language perspective, in the beginning I actually had less features in Rust than I had in Ada that I needed. But like the change was not because of the language that I stayed was because of the language in the community, but the actual initial change was completely based on the tooling that was not giving me the ability to be effective.

Interviewer
As time has gone on with Rust, how do you feel about it as a language compared to Ada? Both in terms of I guess the package management system seems like an improvement, but the other language features, do you feel that Rust has moved in a direction that is providing what you need compared to the use cases you've had in Ada?

Participant
We're getting there. Basically the only thing left is subtyping, custom subtyping, so that you can specify like I want an integer, but only between one and 10. That is something that Ada has built into the language and is a core feature that's basically making embedded development and security critical development very easy. That's something that we're missing. That's something that we need.

4:2 We're getti...

Wants Solution

Interviewer
So the next question is a bit more broad as well. What do you use unsafe Rust for?

Participant
Probably every single use case that we have in the language by now, initially mostly to access the hardware. Basically I was avoiding any kind of unsafe optimization or anything else, just accessing hardware on embedded devices. That was the main thing. Recently to implement multithreaded primitives.

4:3 Ba...
Operating & E...dded S
Preference for Safety

Interviewer
So let's start with that first use case. Could you describe with the embedded applications, you mentioned that aside from the cases where it was absolutely necessary to execute the certain calls that you needed, you mostly stick to safe Rust for optimization. Did you face any challenges in terms of conversion between raw pointers and safe references in that context? Or were the interfaces mostly structured such that you never needed to pass memory between the unsafe world and the safe world in that way?

Participant
So we tried to isolate all the unsafe into a single setting at the beginning, like producing safe references to or safe accessors to the hardware. And so all our unsafe usage was limited to taking a raw pointer in the beginning, converting that to a reference and to calling various volatile or atomic operations in those wrappers.

4:5 So we tri...
Atomic Intrinsics
Local, Minimal Unsafe
Volatile Intrinsics

Interviewer
Gotcha. So to clarify, you had a wrapper that had access to the underlying raw pointer and would perform the intrinsics on that representation. And then in cases where you'd expose that to safe Rust, you'd borrow against it. But you'd begin with like a raw pointer that you'd use for each of those operations.

Participant
I took a hardware address that was specified in the hardware documentation, converted that to a raw pointer, and then the reference it to get a reference out of it. And from then on, it was all field references and two structs that were all safe field access.

4:6 I took...
Local, Minimal Unsafe

Interviewer
Gotcha. Okay. So with the unsafe documentation today, how raw pointer conversion is is described, it's very strict in terms of you need to meet these various preconditions in terms of nullability in terms of alignment. Were those ever necessary to reason about for this context? Or was the structure that you use sufficient to ensure that all of the criteria for

avoiding undefined behavior was met when you were doing these conversions?

Participant
Basically, I don't know issues like that, because the hardware is already aligned. And so in order to get a pointer to the right hardware thing, you already have an aligned pointer. You basically, you never had an unaligned memory situation there.

4:7 Basic...
Contract or Invariant

Interviewer
Gotcha. Okay. That makes sense.

Participant
So just the nature of the app, the fact that you're working within that embedded context, sort of rules out a bunch of the typical pitfalls that you'd have with alignment, specifically, you get a funny pitfall, which is the address zero is actually an address where you can store data. So that is something that's funky in embedded that you need to make sure that doesn't affect your Rust program, because Rust doesn't obviously doesn't like that.

4:8 So just the nat...
Different FFI Memory
Operating & E...dded S

Interviewer
Gotcha. Were there any other cases like that where you had certain assumptions about memory in the embedded context that were directly contradictory to what the rules of Rust are?

Participant
Not that I remember. I can pop off my head.

Interviewer
Gotcha. So when you were writing these wrappers, did you ever reason about pre and post conditions, either as you're writing, like, inserting runtime checks or in writing documentation for certain operations?

Participant
Well, we wrote documentation explaining why our unsafe code was okay, but we never exposed any unsafe operations. We we wrapped it everything in safe operations. We were exposing that to students. We didn't want anybody to actually mess with the stuff. So yeah.

…
Documented...ract or I

Interviewer
29 So there was never any burden on the consumer of the safe API is to provide any particular conditions or else those would have been unsafe.

46

47 **Participant**
Kind of the problem is once you get to to this embedded hardware level, as like, you're undefined behavior, or is you're not doing something

4:1...
Contract or Invariant
Documented...ract or I

correctly by the spec of the hardware, which might not affect your program at all. Like it might. It's not undefined behavior in the common sense. It's more like your device will not act as you expected. And that is something where we documented everything heavily, but we couldn't really add any assertion. So there's no way to accommodate for that in Rust at all. It's really just ensuring that you have the right hardware configuration that you're actually running the application on. You could. The problem is, it's mainly a shift of complexity. It's like, if you implement a driver for a certain hardware, you need to follow the rules of that hardware. And writing that you either already write it correctly, or you rely on someone else writing it correctly. But there's no middle ground where you can make the language really help you do that, unless the hardware provider provides you with a structured representation of the hardware that it can consume automatically and generate something from that.

4:11 Kind of...
Contract or Invariant
Documented...ract or I

Interviewer
Gotcha. And do you find that the producers of the hardware are providing you with adequate specifications and reliable behavior, or do you just have to anticipate the worst in every case?

Participant
Well, it's less anticipate the worst in every case. It's more of the documentation is meant for electrical engineers, not for software engineers. And on the other side, if you have generated information, which we have definitely used, we did a big pre, a big part of data in the Linux kernel that we were actually scraping for our hardware. And that data allowed us to generate all these data structures that I was talking about earlier, those wrappers for the hardware, because they had all the information about addresses and everything in there. But in the end, that only made a bunch of very menial work simple. It did not help us with writing drivers at all.

4:44 And on th...
Generation VS Validati

Interviewer
Gotcha. So could you describe a bit more about the second use of unsafe that you mentioned in implementing a multi-threaded application? Or no, multi-threaded primitives.

Participant
Basically, we have a problem in the Rust compiler where locking is not the right kind of work that we need to protect our data that is accessed from multiple threads. If we lock, that means either we use a read write lock, which gives us infinite readers at the same time, but then the writer needs to wait until all the readers are gone. Or we use a mutex, and that means every time someone reads, the thing is completely locked. But we have lots of data structures where it's actually just like append only. So you only add data to it. You never ever delete or modify anything. And that means we can have infinite reading participants. And at the same
47 time, a writer can do some work at something new to it without affecting all the readers. So there's only a conflict between the writers. And for that, I had to implement a new primitive to support some data structures for

4:12 So you onl...
No Other Choice
Unsafe Data Structure

that. Basically, a vector that could be used multi-threaded.

⋮ No Other Choice
Unsafe Data Structure

Interviewer
Gotcha. Okay. Could you describe a bit more about the challenges you faced in implementing that? Like what unsafe that would be different than other multi-threaded primitives?

Participant
Amusingly, using more unsafe made it less fragile. I was trying to avoid various uses of unsafe before. And instead, you use things like Boxes and vectors to abstract reallocations. But at some point, I needed to deallocate memory. So I was using the raw deallocation API. But I did something wrong. And I wasn't even clear on what I did wrong. But in the end, it was simpler to use the raw allocation and deallocation API directly instead of trying to use the existing higher level primitives, mixing them with the unsafe APIs. That was the problem. Because I didn't have full control. I didn't understand how they were allocating data.

4:13 Amusingly, using mo… Easier or More Ergono

Interviewer
Gotcha. So it was easier for you to transition to just using entirely unsafe under the hood, exposing a safe API than trying to use as much safe as possible.

Participant
Exactly.

Interviewer
Gotcha.

Participant
That might have been avoidable. But like from the time and effort I had to put in, it was easier to just do that. And the final result was much clearer to understand and reason about.

4:14 T… Easier or More Ergono

Interviewer
55 Gotcha. Clearer just in terms. I guess could you add a few more details about how you mean like clearer and easier to reason about?

70

71 **Participant**
As an example, I was basically using a vector to allocate the next block of memory that I needed for my data structure. And in the end, in the destructor of my data structure, I was using the raw deallocate API to deallocate those vectors. And now, since I'm using the allocation API and reallocate API directly, it's basically there's three points where I'm using unsafe and they all are using the same like get the size, get this alignment and then feed them all to the raw allocate APIs instead of trying to match up various different APIs that don't quite match up.

4:15 And no… Local, Minimal Unsafe

Interviewer
Gotcha. So in each one of these cases where you use unsafe, the patterns that you're implementing are the same. So that makes it easier for you as someone who's maybe contributing to this project or just examining the documentation or the code base to understand how these operations are safe.

Participant
Yes, how these operations are safe when used together or after each other. Like each operation on its own might make sense, but like it's a combination that, I mean, yeah, that makes all of this.

Interviewer
Gotcha. Gotcha. And have there been any, so have there been any like experiences in other projects or other uses of unsafe where you've encountered this? Like, is it typical for you that going with this approach of avoiding mixing your different primitives makes things easier? Or has that been like a new experience with this?

Participant
That depends on the APIs that are exposed. Basically, like if there's a good safe API that people have written, I will always go with that. And if that solves the job, I won't even try to use unsafe. It's mainly if I know there's a certain point where I need to use unsafe, I try to use some consistent API or an existing wrapper that makes that doesn't wait with it for me.

4:16 Basically, li...
Preference for Safety

Interviewer
Gotcha. Could you describe a bit about interfaces that have been safe where it's been difficult to use? Like in this case, I think you mentioned using Box at one point for the implementation of this vector, but having problems with mixing that with unsafe, like what are the characteristics of a safe API that make it challenging to use in an unsafe context?

Participant
So when you allocate with something like a Box or an Rc, you don't actually know if they're going to feed the alignment of your type to the underlying allocator or if they feed a larger alignment to the underlying allocator. You don't have control of that during the allocation. So when you manually deallocate, you need to specify the correct alignment. And if you don't know what that is, you can't even do this correctly at all. So if there's too much abstraction, you lose the information that you need to make it actually sound.

4:17 So when you allo...
Arc<T>/Rc<T>
Box<T>
Easier or More Ergono

Interviewer
Gotcha.

Participant
So in that case, you rely on unsafe so you can just access the information that you need to perform manually deallocation.

Interviewer
Gotcha. Are there any other situations aside from alignment where there's some characteristic that tends to be hidden by a safe API that you need access to?

Participant
Yeah, specifically about allocators or?

Interviewer
I guess in any context, not just some equivalent attribute such as alignment that you've been unable to access to the level that you need. So then you end up using unsafe.

Participant
Definitely. And that is accessing bits of a pointer that irrelevant due to alignment or some other information. That's basically tagging pointers. It's impossible to address it correctly or safely right now. Which kind of brings us back to the subtyping of the data has addressed us.

4:18 D...
No Other Choice

Interviewer
Gotcha. Okay. So you mentioned using Box, UnsafeCell, Cell, and RefCell in unsafe contexts. We already talked a bit about Box. So could you describe some of the situations in which you've needed to use types like UnsafeCell or any of the things derived from it and the different challenges that you faced in using that correctly?

Participant
That goes back to the embedded world. When we started using interrupts, we needed to synchronize our accesses to shared memory between the interrupts and the main program. And for that, we needed to implement our own mutexes because there's no operating system or anything interesting there that you could reuse. And for implementing those, we needed UnsafeCell to be able to have interior mutability and then figure out some locking scheme around that to soundly access that shared memory.

4:19 That goes back t...
Data Structur...and Op
No Other Choice
Operating & E...dded S
Unsafe Data Structure
UnsafeCell<T>

Interviewer
Has use of RefCell ever been a performance issue for you versus just using unsafe cell?

Participant
No. So if we're in a single-threaded environment, I just use RefCell

⋮
Profiling

liberally. And I have benchmarked a few cases. And it's usually just a single bully and lookup to check if you can access it most of the time. So yeah, we haven't seen any performance problems with RefCell at all.

4:20...
Profiling
RefCell<T>

Interviewer
Gotcha. Have the semantics of UnsafeCell provided any challenges or any different things to consider for you in terms of memory safety when you're using it in this context?

Participant
Around constant memory or static memory. So I'm working on the [name] for Rust. And so I'm seeing it from the other side. Basically, that's where my challenges have been, not in actually using it, because I've been trying to guard against misuse so much at this point. Around constants, I believe I know all the edge cases. So yeah, mainly from the other side, trying to make the compiler understand UnsafeCell better.

4:2...
Rust Project Contribut

Interviewer
Gotcha. Could you describe some of the cases where some of these edge cases are some of the problems that you've had to guard against?

Participant
So one funny type that Rust allows in constants is, for example, Option of Cell of something. And if that isn't a constant, and you had a constant that is sum and then the Cell, you could technically mutate that with the set operations and so on. Like if you put in a static, for example, and you have static, that's Option Cell, and then you pick the name of static, got set, and you could change it without any thread safety or anything. And that would obviously be a problem. And at the same time, Rust allows you to use that type as the type of static as long as you put none in there. The moment you use some, it stops allowing that. So it's revealing the value of the static to figure out if you're actually allowed to have that value in the static. If you just do the type based analysis, you'd be, no, no, no, that's bad. But since it's compile time, you can actually look at the value and then make the decision. And that has been a can of worms that is amazingly deep. And yeah. So around that, there's so many topics that we could probably spend an entire hour talking about that. So the problem, from my interpretation, it seems like you, from like a static typing perspective, you just never want to allow any use of that without some sort of thread safety guarantee. But what you actually want is every type of static to be sync. So you can access across that. But early in the language, it was decided that we could do some tiny things that are neat, but make it a pain for compiler developers.

4:24 So one funny type that Rust allo...
Cell<T>
Option<T>
Static Variables

Interviewer
Gotcha. Gotcha. Is there another example that you have of a challenge regarding using UnsafeCell statically or even just like not a static context necessarily with something else that you faced in working on the [name]?

115 **Participant**
A little bit the opposite, I guess. Static mut is a horrible other situation where we can't actually make it safe. Like static mut by itself is probably unsound whenever you use it. And the correct way would be to use normal static and build an abstraction on top of that was with UnsafeCell that allows you to access the underlying data, even unsynchronized if you want to do. But as long as you're sure it's sound, it's okay. And there is a small nuance difference between using static with UnsafeCell versus using static mut. And that's basically just that you could have multiple mutable references to the same memory location at the same time. And that's not allowed by the Rust memory model. And yeah.

4:23 Static mut is a horrible...
Static Variables
UnsafeCell<T>

116

117 **Interviewer**
Gotcha. So in the case of static mut, it would be allowing that. But if you make it constant, then use interior mutability, then it avoids that problem.

118

119 **Participant**
Exactly. Because then you can run around with an immutable reference to that RefCell wrapper and only access it mutably when you actually need to. So you never have two mutable references to the same memory location.

120

121 **Interviewer**
Gotcha. So you mentioned using foreign functions in certain situations, which you mentioned are written in C and C++. And then you've used cbindgen and wasm-bindgen to create bindings to those languages. Could you describe, I guess, first, the experiences that you've had with those binding generation tools? So what's been great about using cbindgen and wasm-bindgen? What hasn't been great? What could be improved?

122

123 **Participant**
So the great thing is simply that it works really quickly. You get access to your C API and you don't have to put a lot of work into it of having a somewhat nice API. You don't have to synchronize the implementation on the C side with the implementation on the Rust side. Instead, you'd get a Rust type error if you used it wrong. So you have a single specification. That's the header file. And that is just super convenient compared to writing it once on the Rust side, writing it once on the C side and hoping to get it right. So yeah, just taking off a lot of menial work. What's not so great is that there is ways to write a nicer API on the Rust side that directly is a function interface to the C side without having intermediate layers. Like you could write something that passes a reference to a C function, even if that C function obviously can only declare a pointer. But on the Rust side, you could declare that function interface to have a reference, even with a lifetime linking it to the return type. And if you use bindgen, you need to manually write a wrapper around that that calls that C function. So it's probably more of an aesthetic thing, but yeah, would be nice if that could be done better.

4:25 So the great thin...
bindgen
cbindgen
Generation VS Validati

4:26 So y...
Generate Bind...Fro

4:29 What's not so great is t...
bindgen
Limitation of Bi

Interviewer
Gotcha. So are you in this case having to reason about lifetimes yourself when writing those wrappers and the inputs and outputs for those functions? Or do you have a heuristic that you use in doing that?

Participant
Oh, no, you have to reason about it completely yourself. You need to look at the the code and figure out how those pointers relate to each other and then write those lifetimes manually.

4:27... Contract or Invariant

Interviewer
Gotcha. Have there been problems that you've encountered with doing that correctly? Are there certain situations where it's particularly difficult to write those yourself?

Participant
At the moment it's not obvious, we just resort to keeping raw pointers around and not even trying to abstract it nicely at that point.

4:45... Difficult to Encapsulat

Interviewer
Gotcha. Gotcha.

Participant
So trying to avoid the problem entirely in hopes that in future, there will be a better method for doing this correctly, but it's better for you to just use raw pointers than attempt to write findings that are applying lifetime semantics to foreign code.

4:28 So t... Wants Solution

Interviewer
Gotcha. Okay. And then with, I guess, has that been something that you've encountered both with? I guess the one question that I had was you mentioned only calling functions from Rust written in C and C++, but then you also used Wasm bindings. Are you compiling C and C++ to Wasm and then calling that from Rust?

Participant
No, that was actually a separate project. What we did there was write or use WebAssembly as a kind of high-level language that we converted to a custom assembly language in order to be able to write for that assembly target in Rust or other languages that compile to WebAssembly. And the main issues that we had was that from Rust, we don't have the primitives that we need from WebAssembly. WebAssembly has things like value ref types that basically like shared references or something that you can't convert to an actual address. It's like an opaque address, basically. You can use it, you can pass it around, but you can never actually write it like access it's bytes. And we needed that. And the

4:30 WebAs... Generation VS Validati

only way to get that was either writing manual WebAssembly wrappers ourselves or using Wasm-bindgen and hoping that the optimizer is powerful enough to actually get us to the point that we need.

4:30... Generation VS Validati

Interviewer
Gotcha. Powerful enough just in terms of performance or correctness?

Participant
Not just performance. Yeah.

Interviewer
Gotcha. Okay. Were there any issues there, I guess, both with wasm-bindgen and cbindgen in terms of undefined behavior due to the interaction between language or each sort of compilation?

Participant
Various times, but they were all pretty straightforward where we just realized that it wasn't even a tool that was wrong. It was literally just like we were using it wrong, not using the tool wrong. We were using the API to the tool generated wrong. So it was never an issue with the tool actually doing, yeah, generating bad code.

4:31 Various... Difficulty with Generat

Interviewer
Gotcha. Are there cases where the tool could... So the issue here is not the code the tool is generating, it's just that you're using it in a way that is incorrect or that causes undefined behavior. Is there a particular pattern to that that you found where there's some aspect of the bindings that are generated that might lead you to use them in a certain way that would cause undefined behavior or is it sort of a case by case basis?

Participant
The one pattern that we've noticed was just handling the differences between mutable references, immutable references, and owned references or Boxes, I guess. But whatever other kind of allocator you have in the background, just handling the ownership differences in a C foreign function interface is challenging. It's already challenging just writing C code, but then mixing it with Rust, which has a lot of
139 assumptions built in, you need to actually make sure everything is correct and maybe you have a bunch of pointers that it'll all work out. So yeah.

4:32 The one pattern tha... Different FFI Memory I

152

153 **Interviewer**
Gotcha. So it's the fact that Rust has particular... So is it that you're in both contexts, you have to reason about this type of thing, but then it becomes more challenging because Rust has a particular set of concrete assumptions that you would usually have a bit more flexibility with in C?

154

155 **Participant**

Well, one thing that you have in C is if you forget that the pointer is owned, all you get is a memory leak. And Rust, it gets dropped at some point. And at that point, it'll either... If you didn't have an old pointer and you tried to put it in a Box, then you deallocate something that you didn't have ownership of, that's bad. And if you do it the other way around, that's not really an issue.

4:33 If yo...
Box<T>
Box<T> Bug
From Raw, Wrong Poin

Interviewer
Gotcha. Have there been cases of this in particular with, I guess, aside from Box, just with mutable and immutable references in terms of violating the aliasing XOR principle?

Participant
Yeah. We've had a bunch of these. And the thing, the main issue we've had was they were happening because the programs that we used were too big to run in Miri. At the moment, we had small libraries and so on. We could just feed it all onto Miri and write some mocking on the C side to check that everything works out. And with more complex programs where you get like a 50,000 lines of code C++ space, that's just not possible.

4:35 Yeah. We'...
Borrowing or Provenar
Miri doesn't s...ort this
Stacked/Tree...rrows Vi

Interviewer
I was going to say because, yeah, I've at least read and now starting to observe myself like a 7,000 times slowdown in Miri, just with...

Participant
Oh, it's not even the size. No, literally, because it's impossible to run the C++ code. So it's just going to break your ... We can test the foreign function interfaces with Miri because you can write external functions that you just write a mocking crate and link it against it. And then it'll complain if you screwed up the API there. But yeah, doesn't work for actually invoking lots of C code.

4:34 Oh, it's no...
Miri doesn't s...ort this

Interviewer
So your strategy has just been to create these mocking crates that mirror the functionality of your C++, at least the... Do you have a particular focus on writing these mocks? Is it just attempting to capture the semantics of ownership to ensure that what you're doing there is correct? Or I guess, how complex are the mocks that you're writing and what are you looking for?

Participant
Yeah, they're very basic. It's like most of the time, they don't actually do anything interesting. Some of them just don't do anything because they don't have return value. So we just don't have to implement anything. It's mainly testing the API and then seeing how far we can get.

Interviewer
155 Gotcha. Interesting. There's one other question I had. This might come

back to the embedded system stuff. You mentioned using intrinsics from Rust. Was that usually just within that case? So you're executing these intrinsics on the raw pointers you have under this wrapper? Or are there other situations where you use intrinsics?

Participant
I use intrinsics because I develop on the standard library sometimes itself and that one built on intrinsics. The only other intrinsics that we've used were the volatile ones for the embedded project.

4:3...
Volatile Intrinsics

Interviewer
Gotcha. And have there been memory safety challenges so far that have been unique to those intrinsics that we haven't covered in other through other types or other situations?

Participant
No. I think the intrinsics were chuckling change loss style enough that we could care to not use them too much and also just build good wrappers around them that were simple enough to verify the hand. We've never had any problems with intrinsics as far as I can remember.

4:46 No...
It's not a problem

Interviewer
Gotcha. So this next question could cover any project that you mentioned. Describe a bug that you face that involved unsafe Rust focusing on how it manifested, how you pinpointed the cause, and then what the fix was.

Participant
I mean, I already described the thing earlier with that library where we did the alloc stuff, but I don't have a good example beyond that right now. I can do a general thing, but yeah.

Interviewer
No worries. Yeah. The earlier example you had was excellent. So you also mentioned using both Clippy and Miri. I guess we've talked a bit about the challenges with Miri in terms of having to deal with large foreign code bases that it just will not function on. Could you talk a bit more about your experiences using Miri in Clippy when the tools have worked?

Participant
I guess over like a pure Rust code base, maybe. The main thing with Miri was always introducing it later in the project. I mean, I guess it's the same with Clippy. If you introduced either of these tools later in the project, it becomes much more challenging to run your test suites with that or run it across your project. If you started from the beginning, you always making sure that already works with those tools. That's been the biggest hurdle
169 that I've had so far with Miri. Again, since I'm working on it myself, most of the time when I had an issue, I just fixed it myself. So I didn't have that

4:38 The main thin...
Difficult to introduce
Miri is slow

many issues that we're blocking. That's probably not representative. With Clippy, the issues are upgrading Clippy to a new version that causes lots of more lints to get triggered and simply using it on an existing code that hasn't been run with Clippy a lot. There is no good tooling for gradually integrating Clippy. You can only manually slap a bunch of allows if you were. But I've seen tooling for similar static analysis tools where you'd run it and then it would just store all the information about all the lints that triggered in a separate file and then you could gradually just work on it to get rid of them or address them in some manner. So having that approach with Clippy would be considerably helpful for these projects where you have to introduce a little later stage. Exactly. And also for upgrading Clippy because that's basically the same as introducing it later. Yeah. I guess with Miri because you're dealing more with patterns of using ownership and mutating through certain references.

4:39 With Clippy, the issues are...

Difficult to introduce

Interviewer
Have there been patterns that you've seen when you've introduced Miri late on and then you suddenly have to deal with a bunch of these errors from it? Are there typical errors that tend to go undetected for a while that you then have to deal with when you introduce Miri or is it just really dependent on the project and there isn't a clear pattern.

Participant
It's more just everything that Miri detects it will detect the moment you start running it. No, there's certain patterns. One thing we've seen is that using synchronization primitives wrongly or using multiple mutable references to the same data keeps happening and is not unsound in practice. So people will have been using that left and right and basically they're using multiple mutable references to the same data and not using the correct synchronization accesses, like not atomic operations or nothing, but sometimes they use for example volatile instead of atomics, which is still not correct, but it's like making sure that the optimizer is sufficiently restrained to not break your code in really bad ways. So it could happen at any time when you do an LLVM upgrade or something, when you get to a new Rust version that your code suddenly breaks, but it is very unlikely, simply because the code is so far apart that the optimizer will never see both things at the same time and realize that you're doing undefined behavior and that it can go off of the rails and just do whatever it wants.

4:40 One thing we've seen is that using synchr...

It's not a problem

Interviewer
183 Is there a particular way of reasoning about this where you can determine that something or you're sort of breaking the rules, but I guess the way you said it was not unsound in practice, I guess, how do you know when something is not unsound in practice?

190

191 **Participant**
You don't. You know it basically by, or you're guessing at it by your program is running fine on the millions of machines or something and so if it's not stack folding on those machines, statistically it looks like you're

4:47 Y...

Tacit Knowledge

not unsound and that might be true for your specific compilation and the binary that you're producing, but it might, it's unclear when that will break. So you really just have to continue testing and ensure that these, that the behavior is still consistent, but there aren't any good formal ways of going about doing this. I mean, that will work to a certain point. The problem with software is always the breaking point is not, it's not like, well, what's the example like, like steel beams, it won't bend and then break, it'll just break randomly. And if you have an attacker who's sufficiently knowledgeable about your software, they will break it. So having used cases like that, it is only sound in practice because you don't have a sufficiently motivated attacker. But once you have someone with the motivation and the knowledge, then that becomes a problem. Yes, definitely. So in this, oh, sorry. Miri is just a really bad fuzzer. It's like, it's not a fuzzer, it does it done mystically, but it finds the same things that a fuzzer does. So if somebody uses Miri on your tool and finds the situation where something breaks, they can then target a fuzzer for that exact input and find an actual attack vector. So, yeah.

4:47 You do... Tacit Knowledge

Interviewer
So have you seen that pattern used against you, I guess?

Participant
No, not really. This hasn't happened. This is just a theoretical attack vector that we're concerned about and trying to avoid, obviously, by running Miri on our tests. That's it.

Interviewer
So I guess in this case, even though you don't, like you, so you aren't ignoring stack borrows or tree borrows violations. It's just that a lot of times that might not be in sound in practice. Or is it the case that those are all the things that you'd correct immediately?

Participant
There's two things. There's one where Miri just doesn't detect it, because it doesn't happen in the executions of our tests. But that's the rare case. The usual case is actually we detect something and it's way too big to refactor. And getting there is like a multi-year project. So it's happening in the background. We're ensuring that not more issues are cropping up, but we know which tests are not going to pass and we need to work on them. And yeah, we know about the soundness issues, basically.

4:... Missed a Bug

4:42 The us... Laziness Fights Toolin

Interviewer
Gotcha. So to summarize, you are running Miri on these tests. You know that it isn't necessarily complete coverage, but that's okay. And you are seeing cases where you might get a borrowing violation at runtime from Miri. But in practice, it's both a lot of investment to be able to fix that. And a lot of times these issues are not unsound in production. So it's okay to have to live with it for a bit while you refactor. Gotcha. Okay, interesting. So you mentioned a couple problems today where it seems like these

191

tools aren't quite doing what you need them to. The ones that I'm thinking of are with binding generation tools. It would be great if you had more automated generation of wrappers, like being able to create lifetime contracts for these bindings or something you mentioned. And then also with Miri, there seems to be sort of two problems that you pointed out, one of which is coverage in that Miri is only as good as your test cases are. And then also in compatibility, and at the moment you get these large C++ code bases interacting with Rust, it's just no longer something that's useful to you. Are there any other problems related to the development tools that you've talked about where it's just something that you really wish it could do? Or there's something that's just undetected that you see as a problem and that you really wish that these tools could help solve for you with unsafe?

Participant
I mean, one thing has already happened. I always wanted Miri to be able to execute functions without knowing the arguments. Basically symbolic execution and then running a sat solver to find out if there's any problems in that function. And turns out people have developed that. There there's multiple tools in the Rust community in the data analysis community here that have demonstrated that they can do that for certain simple functions, simple being a very stretchable term. So some of these are actually getting very complex and they detect unsoundness without ever actually executing the program. There are cases where the solver will just get stuck forever and never prove that your code is correct. But yeah, and it finds bugs real bugs in real code. And so yeah, people have developed that and it's getting usable. So I guess somebody solved that problem for me. And I don't have to do it in Miri actually closed the issue just two or three weeks ago.

Interviewer
So I guess, do you have any opinion on these approaches? Like, do you feel that they would scale to the use cases that you described with large scale like interop between C and and and Rust or?

Participant
No, this is for completely pure Rust for now. Or going to with interop, the thing that we're exploring is basically a Valgrind version of Miri where you could run your program without modifying it without running without compiling it especially for Miri or something. What you'd do is you'd feed it to Valgrind and Valgrind knows about the Rust rules, like with a certain plugin that we've been developing. And then that will also uphold your constraints within the C code base. Treating the C code base basically as a Rust code base that is just raw pointers everywhere and unsafe everywhere.

4:43 No, this is for compl...

Wants Solution

Interviewer
Gotcha. Okay.

Participant
So yeah, having sort of like, gotcha, so just yeah, extending the current capabilities of Miri not necessarily like no changes to the semantics of the checking just implementing a different tool so you get more competitive.

Interviewer
Awesome. Any other problems you can think of?

Participant
Not right now.