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

Participant
I guess it's just kind of the best tool for the job for a lot of the things that I do, and I really enjoy a lot of the ergonomics and the features that it provides. It's the niceness of using a functional programming language without a lot of the baggage that can come along with it.

11:42 I... Rust is ergonomic

Interviewer
Gotcha. So I guess what use cases then do you feel that Rust is a good fit for that you usually encounter?

Participant
So that's kind of two different questions. That's, I mean, the more broad question of what is Rust good for versus the question of what do I personally use Rust for. For the broader case, I mean, it's a general purpose programming language. So in that, I would say it's generally useful, generally applicable to a lot of problems. For me specifically, a lot of the stuff I do is with language and compiler design and databases and such. It's very nice to write high-performance database code in Rust and know that it's not going to explode on me, and it's also really, really nice to write compilers in Rust because of things like some types or enums and all that. Yeah.

11:1 a... Databases

Interviewer
Gotcha. So is it mostly, so it's a combination of performance and safety then, would you say, as well as just more functional features that you find are useful for you, I mean?

Participant
Yeah. Yeah. Ergonomics plus safety plus speed. Yeah.

... Memory Safety
Rust is ergonomic
Rust Performs Well

Interviewer
Gotcha. Okay. So then what do you use unsafe Rust for?

Participant
I mean, the less safe things, you got to squeeze out the little bit of performance. A lot of auto vectorization, or sorry, vectorization, the occasional data structure, like lower-level data structure, let's see, I mean, currently I'm writing a JIT compiler, which involves a lot of unsafe, FFI, that sort of thing. Yeah.

11:... Increase Performance
11:3 I... Compilation & Interpre

Interviewer
Gotcha. So let's start with the performance then. What are some of the cases where you typically employ unsafe to improve performance of

something that would usually be safe?

Participant
Like, or what criteria do I use to judge whether I should or shouldn't do that? I mean, or?

Interviewer
Yeah. Sure. I mean, that, I guess, just like, what are the situations where you just in general employ unsafe to improve performance?

Participant
Yeah. Well, I mean, generally speaking, it's like on a problem point that we've interacted or we've encountered or identified. And so, generally speaking, it's backed up by a benchmark of some side. I'll totally admit that, like, sometimes it's more of a, like, I think this is going to be a hot thing. So I'm going to kind of prematurely optimize. Probably not the best habit, but realistically speaking, it does happen.

11:... Profiling

Interviewer
Gotcha.

Participant
Yeah. Yeah. I mean, that's most of the case, or like data structures that you can't really express without it.

11:... No Other Choice

Interviewer
Like which types, and sort of to clarify, you generally do have performance numbers that support a particular decision to use unsafe.

Participant
Yeah. Yeah. I try to. Yeah.

Interviewer
And then for data structures, is that just like cyclic patterns? I know that that's kind of the almost stereotypical unsafe, rust case or under particular.

Participant
I mean, generally speaking, or not for me, no, I end up writing a lot of like arenas and arena style things, because I'm like, you know, for one reason or another, specializing something to address a specific task. And you know, like, indice-based arenas, I've done them plenty, but sometimes you need like a pointer-based one. Like another recent data structure I've written was basically like a dynamically dispatched vector. It was like a VEC that held one V-table pointer in heterogeneous elements, and yeah, I mean, yeah, that sort of thing.

11:6 I end up... Custom Allocator
Pointer Arithmetic
Unsafe Data Structure
11:7 Like... Unsafe Data Struct

Interviewer
Gotcha. Gotcha. Okay. So cases where, okay, so arenas and then that particular custom Vec type that handled the situation that you encountered. And then, all right, so you had performances use case, you had implementing data structures as a use case. And then a lot of your work revolves around JIT compilation and the FFI mentioned. Definitely two use cases where unsafe will be involved broadly, I guess, starting with the FFI, like what generally, like what use case do you have for the FFI?

Participant
Like what do you mean?

Interviewer
I guess, so is it that you're someone who tends to create bindings for existing libraries written in other languages and then make them accessible for Rust? Are you working with the FFI and that you have like a mixed Rust and C code base for both or like evolving concurrently then? I guess just trying to narrow your, like how FFI is useful to you in the types of applications that you're developing.

Participant
Yeah. Okay. So on the JIT side of things, FFI, at least in my current stuff has been used relatively broadly. I mean, like, I don't know how much you can call it like FFI, but, you know, when you've like generated a function, you have to like dance around calling it. Yeah. That's like FFI to one extent, but also like I end up like having intrinsic functions of sorts that I'm calling from within JIT code. And so that's, I guess, an FFI of sorts. We also, for work, we've had like a few consumers in other languages and so we've written bindings for that, Python bindings that kind of went through C or in C bindings and that sort of thing.

11:... Intrinsics

Interviewer
Gotcha. Gotcha. Yeah. I was going to say from your, from the survey responses that you've worked with pretty much almost every language that there is a foreign bindings support for and in rust from, from everything that you selected.

Participant
So yeah. Toyed with a lot of languages.

Interviewer
Gotcha. Would you say that there are a few that are ones that you typically work with?

Participant

Um, Everything. So it depends on whether you're meaning like seriously or, or not, I guess, because currently for work, I'm, it's almost entirely in rust. There is some job that I do have to interact with, but I don't do it a ton. That's not really my job, but, um, most of my like other language consumption is kind of on my own time because I do it because it interests me. And so it's just more of like a, you know, let's learn this language for a week or two in my own time just to like get your grips with it. You know, why do people like it? What are the interesting features I can take from it? That sort of thing.

11:45... For Fun

Interviewer
Uh, okay. Gotcha. Um, so rust has these assumptions about reference types and, uh, a fairly strict aliasing model for both it's safe types and, uh, also just in terms of what counts as undefined behavior when you're using raw pointers, how do you reconcile the way that rust reasons about memory with how each of these different languages reasons about memory? Like what are the challenges that you've faced and say reasoning about how, uh, a rust reference is lifetime is handled when you're passing it as a raw pointer into, into some other, um, other languages on time.

Participant
Um, I mean, generally I don't. Gotcha. Uh, like, I mean, I try to avoid situations like that because it's just kind of setting yourself up for pain and suffering in the, in the longterm. I mean, truly speaking, I try to keep things as, I guess, limited as possible to where you, you know, like, allocate a Box, call a C function that doesn't like capture the pointer and, and then, you know, deallocate the Box after, you know, that sort of thing, like keeping the scope as, as minimal as possible. Like, you know, I mean, it's definitely, I mean, I definitely have done stuff like I've written LLVM bindings, which was not a fun, not a fun task to where essentially you've got long lived references flying around and it's not fun to deal with. But that's more of a, I wouldn't necessarily say that's kind of like an inherent rust problem or even really a rust problem at all. That's more of kind of, in my opinion, like a symptom of how C plus plus does memory management.

11:10 I try to avoid... Allocation in Rust
Simple FFI
11:13... Box<T>
11:... Local, Minimal
11:11 I... Difficult FFI
11:47 But... Different FFI Memory

Interviewer
Okay. So if you're working with a large scale C plus plus code base, it would be more likely for you to encounter that problem. Like, is it, is it just generally a C plus plus thing unless of just the way that LLVM is crafted?

Participant
Um, so I mean, LLVM is definitely a really, really weird, like, I mean, it does a bunch of incredibly sketchy stuff and runtime type information and yeah, I mean, yeah, I don't know, honestly, um, like, I mean, a lot of sufficiently large C plus plus code bases turn out really kind of cursed editions, but yeah, I don't know. I don't know. Honestly.

11:48 LLVM i... C++ is Difficult
Different FFI Memory

61 **Interviewer**
Gotcha. So like, just one of the things you're getting at earlier, um, and just make sure I'm following correctly, it's that you attempt to engage with the FFI in a way that limits your, like the degree to which the memory models are interacting to avoid undefined behavior, but then there are some cases like with the LLVM bindings where that's just going to be necessary because of how the library is working.

62

63 **Participant**
Yeah, essentially. Yeah.

64

65 **Interviewer**
Okay, gotcha.

66

67 **Participant**
Yeah, kind of, kind of minimizing the liability to some extent. Yeah.

Local, Minimal Unsafe

68

69 **Interviewer**
And then when you are, I guess, minimizing the liability, one of the things you mentioned was the pattern where you allocate memory in a Box, unwrap it as the raw pointer and then pass it through and then at the end, rewrap it and have it deallocated.

70

71 **Participant**
Right. Yeah. Yeah. Why would I just use that as kind of an off example, but yeah, yeah.

72

73 **Interviewer**
Um, I know one of the things you mentioned there that was interesting is that you ensure that that pointer is not copied or stored on the, the C side of things. Um, that seems like something where, like, are, are you relying on your own knowledge of the code bases to be able to determine that? Like how easy is that to audit?

74

75 **Participant**
That's, that's generally your only really option. I mean, there's no way to really automate that check of that a, you know, that an arbitrary C function doesn't capture your pointer. I mean, generally speaking, most bindings or, you know, functions will be relatively sane with that sort of thing, but yeah, there's, I mean, there's no really way to guarantee that unless you're doing some really deep, like, I don't know, you can probably verify it with a live or something.

11:50 I mean, there...

Difficult FFI

76

77 **Interviewer**
Gotcha.

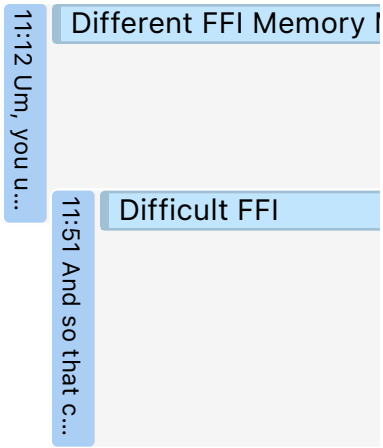
78

79 **Participant**

So then yeah, that would be very difficult to audit, I think, or it's not audit, but to like machine prove.

Interviewer
Ah, okay. Are there any other properties like that that you reasoned about when you're working with data pass across the FFI?

Participant
Um, you usually threaded this is a big issue because a lot of C code bases are not, or like C and C plus plus code bases are not very transparent about their, their behavior in the presence of threading. Like even, you know, lib C is kind of notorious for how terribly some random, seemingly innocuous functions function in the presence of multi-threading. And so that can be a very difficult thing of trying to ensure like, you know, or trying to even figure out in the first place, like, can I pass this, you know, my like LLVM handle across threads or is it going to behave horrifically if I do that and trying to like actually figure out whether or not that's an okay thing to do in the first place can be really difficult because some like some, or a lot of code bases don't even like say it in the first place.

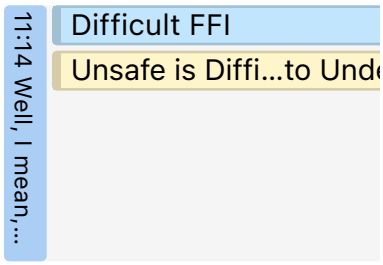


Interviewer
Gotcha. Are there any, so it's the notion of whether or not your pointer is being stored, whether or not it's safe to, I guess with the LLVM handle you mentioned, that's something that's coming from C into Rust then. So right?

Participant
Yeah.

Interviewer
So it's whether or not you can treat that thing as thread safe in a Rust environment coming from a C environment, right?

Participant
Yeah. Well, I mean, really, I guess kind of the core or most of the issues I've had with FFI are trying to figure out or it is the fact that it's really difficult to figure out a lot of the properties of a given code base. Like it can be very difficult to figure out like, you know, are you actually mutating something? Are you retaining pointers to this? You know, like if I give you like some input data, are you going to retain a pointer to that and keep operating on it later? You know, are you referencing it?



Interviewer
Gotcha.

Participant

Kind of thing. You know, are you sound in the presence of threads? Do you use thread local variables? Like how do you actually behave? It can be really hard to figure out sometimes, which, you know, is annoying if you're trying to make like a general purpose library or, you know, like kind of a library layer on top of that to where you have an ergonomic Rust interface. And it behaves in all of the ways that you would expect a Rust interface to in a sound and all that fun stuff. But yeah, it's a lot of the times it's a very human issue of lack of documentation and lack of like correctness within a code base.

11:52 You know,... Difficult FFI

Interviewer
Gotcha. Okay. Yeah. Yeah. So and you were doing your best job in terms of just analyzing the source code yourself, just and auditing it to determine these properties, correct? Like you don't have any mechanisms or patterns that are helping you. It's just your judgment of the properties of the C code and how you write those bindings.

Participant
Yeah. A lot of like reading documentation, trying to ask people things, that sort of thing.

11:... Engaging with...st Com

Interviewer
Okay. And then I guess surrounding. So a lot of what you're describing seems to be functional correctness of the code, whether or not you're actually accessing memory in the correct way. I know one other issue with foreign bindings can be the types that you declare on either side. Like, I guess the thing I'm thinking about is the Rust compiler has typing lints for, like if I just try to pass a Rust string as an argument to some sort of foreign function, it's going to yell at me because that type is just incompatible. Like I can't declare a C function that takes in a Rust string. But then I guess there is some gray area where there are some types that Rust might warn you about, but you can still technically like pass in and they'll be interpreted. And then there are problems that might happen if you like have slightly different types declared.

Interviewer
I guess has, have you had issues in that sort of area where the declaration of your foreign bindings causes some sort of problem?

Participant
I mean, yeah, that definitely is like a very real issue. I guess to, for, to some extent, I guess I've kind of learned of just like be or learned to kind of be as conservative as possible in regards to bindings to where I'm basically only messing with pointers and like, you know, scalar primitives and pointers, you know, like typed pointers, sure. And like non-null and option non-null. For the most part, it's primarily pointer types across the FFI boundary.

11:15 I guess I'v... Simple FFI

95
107 **Interviewer**

Gotcha. So you haven't ever like encountered an improper typing warning from the Rust compiler and had to disable it because you know what you're doing is correct. It's always that you're within the space where the standard of types is like really like defined as being correct.

Participant
Yeah. I mostly avoid like trying to deal with, you know, like arrays and that sort of thing or, you know, non or types with an undefined representation and that sort of thing.

11:54... Simple FFI

Interviewer
Gotcha. Okay.

Participant
Yeah. Yeah. I would assume that's a lot of what people encounter is they're trying to pass like wrapper rust types across the boundary to where you should really, really not do that. But I kind of, I almost can't do that because I'm the person who wrote the randomized layout flag. And so I feel like that would be kind of a bad move.

... Rust Project Contribut

Interviewer
Yeah. Yeah. I know one common pattern at least in our auditing of the, like where we're sort of doing a few studies on that and we found that a lot of crates declare 128 bit integer types. And then have to, as like part of your foreign function boundary, and then the rust compiler is like overly strict about never allowing you to declare a foreign function with one of those types.

Participant
So yeah, I mean, I'm somewhat of the opinion that the rust compiler is correct in that, because I mean, C does not really provide very many guarantees about a lot of things. And so, you know, from the perspective of the rust compiler who's trying to do, who isn't just trying to do like in practice, like in practice correctness kind of, like it's trying to do a total correctness check for everything. And so in terms of total correctness, no, you can't pass a 20 or 128 bit integer across the boundary because, you know, C doesn't support that. You know, some C compilers do, but C does not support that, you know, as a standard, which, you know, is very much the fault of C. But yeah, yeah, yeah.

11:16 And so... Different FFI Memory I

Interviewer
Gotcha. So then I guess in the area of binding generation tools, then you've also used almost most of the ones that I selected. So I guess, could you just describe what your opinion is on binding generators in general? And like, have there been cases where they've been really, really great, really helpful versus situations where the output produced was wrong, or there was some other unhelpful aspect or like a usability concern that was a challenge for you?

120

121 **Participant**
Yeah, I mean, sometimes, I mean, there's edge cases and bugs. I wish a lot of them were kind of better maintained, because a lot of them are kind of very much passively maintained. You know, they're not really like or which like, you know, no fault to the maintainers because, you know, we all have lives and it's hard to constantly maintain big code bases. But you know, it is unfortunate that a lot of them are kind of in the state of like, if you want something, you can do it. Because that can be difficult if you aren't familiar with the code base and stuff. But I mean, by and large, they're like, they're very convenient when you've got a really big surface that you have to cover. You know, it's I mean, it's not feasible to bind all 10 billion functions from, yeah, I don't know, some random C library. Yeah. But, you know, like trying to bind the entirety of WinAPI or like, WinAPI by hand, that would be horrible. I feel, yeah, that would never get done. But yeah, I mean, it's, I mean, they definitely produce raw bindings and very much C style bindings, which is okay. It's exactly what they're supposed to do. And realistically speaking, I don't think a machine would be able to translate or to be able to produce like, rust interfaces out of kind of poorly defined C ones, but yeah, they're tools, I guess, yeah, day to the day.

11:17 I wish a lot of th...

Limitation of Binding T

122

123 **Interviewer**
So you mentioned that you do write the manually sometimes then though, is that in cases where there's just like a smaller API service, or you may need to make like, some tweaks to fix those bugs and corner cases?

124

125 **Participant**
Yeah, or if I've just got like two functions that I need from WinAPI, I can just like, you know, manually write the binding. It's not that big of a deal. But yeah, yeah.

11:18...

Generation VS Validati

126

127 **Interviewer**
So then the next question I have is surrounding memory container types in any unsafe code, just not particularly to FFI. Actually, go back one quick.

128

129 **Participant**
It's very convenient for like header generation. It's very nice to be able to like automatically keep like, you know, your C headers, like, you know, if you're exporting a C interface from a library, it's very, very convenient to have it just like automatically be updated. Super nice.

11:56 It's...

Generation VS Validati

130

131 **Interviewer**
I see. So if you're using something like cbindgen?

132

133 **Participant**
Yeah, yeah. It's incredibly convenient. Yeah.

...

cbindgen

134

135 **Interviewer**
Okay. Gotcha. Yeah. So you're, are there any development challenges that you feel actually started to follow up, I guess, that I thought of? Are there challenges you feel that are unique to one direction of FFI binding to the other, like calling something that was written in Rust, but that you're like calling from a C code base versus calling something written in C from Rust?

136

137 **Participant**
I mean, generating like, or, I guess, other language to Rust bindings, probably one of their biggest difficulties, because like for the most part, it's probably pretty clearly defined, unless, or, you know, like, things like PY03, like it's a pretty defined interface that they're making, I guess, probably the biggest difficulty on that side would be like, how do we translate Rust or like the given Rust documentation into something that's sensible with or for another language? Or how do we give the user the ability to create sensible documentation?

11:58 I gues...

Idiomatic FFI Encapsu

138

139 **Interviewer**
Gotcha. Okay.

140

141 **Participant**
You know, because like a Python consumer of a library, doesn't really care about how you call a function from Rust. And the Rust examples are irrelevant to it, or to them, but they still want document or should have documentation of some kind.

142

143 **Interviewer**
Gotcha. Okay. But yeah, yeah. That makes sense. So the next question that I have is about memory container types. And since you selected every single one, I'm going to go ahead and list it. It seems to be a pattern. I'm just going to sort of explain what the idea behind this, having this question was, and maybe that will like help narrow down a few particular situations that you might have in mind. So I was thinking of cases where you have a type like a Box or an arc, where you might want to access it as a raw pointer. So you need to unwrap it. And then that gives you this additional obligation to rewrap it then to ensure that things are deallocated properly, or otherwise sort of handle that reference count or just the otherwise, like, forget the allocation so it doesn't get leaked. So along those lines, are there any unique challenges that you face when using Rust's sort of safe, primitive memory container types with raw pointers? Or like as raw pointers, I guess?

144

145 **Participant**
Like, sure. Or I don't know. I guess I'm a little bit confused about the word of the question, but like specifically the as raw pointer part.

146

147 **Interviewer**
I guess not. It can be more broad than that, just like to contain raw pointers, accessing them as raw pointers, just I guess, challenges in which you are using these data structures to contain or just like along with unsafe operations.

148

149 **Participant**
Okay. I mean, it requires a lot of mindfulness to interact with collections unsafely, you know, like, I mean, you have to be aware and be, I mean, knowledgeable over the first place. So they're the kind of the constraints that they have, like, you know, you need to not, you know, write past the end of an array because you didn't, or a vector because you didn't allocate enough memory, you know, you need to be mindful of like what operations reallocate or could possibly reallocate that sort of thing, you know, like, are your pointers going to be invalidated? For some things, it means you like, you know, or for, I guess, more FFI-minded things, it means you have to be mindful of like, have, you know, did I offer a, you know, like a decrement reference counter function and an increment reference or like a clone, like, you know, do I provide a C version of like drop RC and clone RC, you know, like that can be really relevant depending on what you're doing? Yeah, I mean, I guess the biggest thing is just mindfulness of like, how containers behave or how a particular container behaves and like upholding its constraints where necessary.

11:21 . So they'r...
Contract or Invariant

150

151 **Interviewer**
Gotcha. Do you have an example of a constraint that is particularly challenging to uphold or one that you've often forgotten to check correctly and had bugs show up because of it?

152

153 **Participant**
I mean, reallocation is a big one. I mean, invalidating previously held pointers can really trip you up if you're not careful, especially if you've written, oh, this is another, I guess, custom data structure, like I wrote a thin string a while ago that stores like the length and capacity on the heap along with the data. And so like, you know, you think it's going to act like a normal string, but you have to be really careful in regards to pointer invalidation because it's very, you know, like, if the operation reallocates the pointer you previously had is completely invalid. That sort of thing.

11:20 this is ano...
Contract or Invariant
Unsafe Data Structure

154

155 **Interviewer**
Yeah. Gotcha. So are the challenges that you face generally with memory invalidation then and not memory leaks? Or do you not see memory leaks as an issue?

156

157 **Participant**
Yeah, I mean, I mean, memory leaks definitely aren't an issue. I guess not as big of a one as like actual correctness or soundness problems.

158

159 **Interviewer**

Gotcha.

Participant
They're usually a little bit easier to address because it just means you generally speaking, that means you just forgot something somewhere. Instead of like, oh, I structured this incorrectly, I used this thing after I shouldn't that sort of thing. It's a little less insidious of kind of a bug.

11:59 The...
Memory Leak
Memory Leaks

Interviewer
Gotcha. Gotcha. Well, so extending on this, what's an example of a bug or just something unsound that you discovered in unsafe code?

Participant
I mean, a good amount of them. I mean, a lot of them of my own creation. But let's see. I mean, the aforementioned like thin string reallocation invalidation stuff. Couple data races. Some little like fun memory ordering bugs with atomics.

11:...
Atomic Intrinsics
Concurrency Bug
Data Races

Interviewer
Oh, okay.

Participant
Yeah, I mean, I mean, probably a lot of things at this point in time. But yeah.

Interviewer
Gotcha. Is there any, I know one of the particularly difficult things can be the access permissions that are associated with raw pointers, like if you take a borrow and convert that into a pointer, it sort of retains some of the same provenance is the term that at least like Miri uses. Yeah. Has that been an issue where just the you've maybe broken rusts invariants on uniqueness or mutability and aliasing?

Participant
So this is going to be a bit of a weird answer because I'm like very familiar with the kind of the aliasing rules and everything surrounding that and just aliasing and provenance as a concept because I interact with it a lot in terms of compilers and stuff. And so I think about it a lot. And so because I like, I mean, I mean, I don't want to like try and toot own horn or anything like that. Like just because I think about it a lot and because I've dealt with it a lot, I generally don't run into that sort of issue. Just because I'm, I kind of think like the compiler in regards to it. And so yeah, yeah. But from my co-workers and stuff, I've definitely fixed plenty of like issues that they've unintentionally written that seem like stupid little nitpicks. But it's, you know, for one reason or another, important to provenance.

11:60 I'm like very...
Tacit Knowledge

11:61...
Borrowing or Provenar
Stacked/Tree...rrows Vi

Interviewer

Are there any common patterns that you tend to see people making with mistakes along the time?

Participant
Usually it's like reference and validation to where you, you know, like, like, or calling like as mut or as a mutable pointer or getting a mutable pointer to a vector and storing it and then calling the vectors length, you know, like that's like a a kind of debated case at the moment because the memory model isn't fully decided. But like generally speaking, like calling a function that later invalidates the reference that you created, you know, or the pointer that you created. Even though like in practice, it's not actually an issue. It's like a kind of a model issue. Yeah. Or an issue on a model level.

11:23... Borrowing or Provenar
Stacked/Tree...rrows Vi
11:... Shifting Ground
11:25... It's not a problem

Interviewer
Gotcha. Gotcha. Speaking of models and just the general architecture of un-safety, could you describe the extent to which un-safe is used within like, in particular, I'm thinking of the JIT code. Are there any architectural decisions you make with un-safe and like just what's the distribution of un-safe in those types of code bases where you're working with?

Participant
It's pretty, it's pretty horrific to be honest. Like, I mean, I probably need to write some more abstractions for it because like as is, it's, it's not great. It's just basically like, like where all of the interaction with the JIT or like, I guess the produced code is, is basically just one big un-safe block. So like, not great, because it's really hard to express that kind of thing. And like inherently when you've got like, you know, when you're taking random pointers, quote unquote, out of nowhere, turning in the function pointers that are also unsafe to call, you know, you're just going to inherently interact with a lot of unsafety.

11:26 all of the inte... Extensive, Non-local U
Unsafe is Diffi...to Und

Interviewer
Gotcha. So I guess there are two themes there. One is that you could potentially increase the amount of safeness by maybe factoring out unsafe operations and encapsulating them, but that there are some things at the end of the day when you're working in a JIT that will always be unsafe and that you cannot avoid.

Participant
Yeah. I mean, at the end of the day, like you are doing something very unsafe to where you're doing something that the compiler cannot possibly check. And so you have to take over the wheel and tell the compiler, I know what I'm doing. This is correct. And so at the end of the day, it is going to involve a lot of unsafety. Yeah. Yeah. I think that's pretty much inherent to the problem.

11:62 you... Tacit Knowledge

Interviewer
Gotcha. And how, I know one of the key Rust principles is that you want

an unsafe code to be very minimal and encapsulated, but in particular, like easy to reason about, like in isolation. How well do you feel that that applies to your code base?

Participant
The JIT? That's very difficult. Yeah. I mean, it's a very abnormal case, for like kind of the non-JIT side of the code base. Everything definitely is encapsulated where in safety is involved. And I try to make it as easy to reason about as possible, but it's just like a very atypical case to where it's kind of a pervasive, like trust me kind of thing.

11:27 The JI...
"Safe" API
Exposing a Safe API

Interviewer
Gotcha. So is it like, would you say that part of it is just domain specific knowledge of JIT compilation? Like maybe someone who is like some other developer on this project who has equal or similar experience to you would be able to pick out an arbitrary unsafe block and be like, yeah, you know, I can kind of reason about what's happening here. Or is it just really difficult for everyone, regardless of your level of expertise with JIT?

Participant
Well, it's, it requires non-local knowledge, you know, like you can't just look at a given unsafe block and be like, yeah, based on, you know, the context of which this is called in, this is correct. It's, you know, you have to know the innards of the JIT compiled function. So you have to, you know, you just have to have external knowledge outside of that specific unsafe block, which is, you know, so it's just not encapsulated. It's still like doable. You know, you can still look through like, you know, what code or like what code is being generated. And I say what code is being generated. I don't mean like physically generated. I mean, like, you know, what like JIT instructions have you put together kind of thing. And then from there, no, like, okay, this is going to be, this is correct or not. It's not like a local reasoning, or you can't really locally reason about it. You know, you can't say like, oh, you know, we already checked that the pointer was non-null. Therefore, non-null new unchecked is correct to call here. You know, it's just not a locally encapsulated thing.

11:28 Well, it's,...
Extensive, Non-local U
Unsafe is Diffi...to Und

11:63 It's no...
Extensive, Non-local U

Interviewer
Gotcha. Do you know, like, is the specific JIT project you're working on something that's open source and that you could name, or is it proprietary that you wouldn't be able to in this?

Participant
No, yeah, it's open source. Yeah.

Interviewer
Okay. Would you be comfortable sharing like what what project that is a part of?

202 **Participant**
Yeah, it's [redacted]. Gotcha.

203

204 **Interviewer**
Okay. Sounds good. Yeah, I might take a look at that code base then.

205

206 **Participant**
Yeah, it's, it's like [redacted.

207

208 **Interviewer**
Yeah. Gotcha, yeah, no, that's super helpful. Thanks a ton. Yeah, really appreciate it. So, I guess then let me double check this. So you've used also every development tool that I listed there. I guess could you describe then just the how bug finding tools fit into your development workflow with Rust?

209

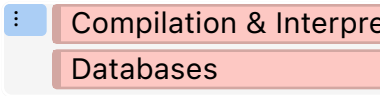
210 **Participant**
Yeah, I mean, Miri is incredibly helpful, you know, like it's a great diagnostic tool. Usually for a lot of them, I end up or I guess more of the the checkers, you know, like all the sanitizers, Miri, that sort of thing. I usually run those as part of CI or usually part of my or for some things part of my test suite, if it's, you know, like it specific enough. But I'm a Clippy advocate. I'm a Rust FMT advocate. I use them extensively. I think they're fantastic tools. I think it's great that they help enforce consistent codes, code styles across not just code bases, but entire ecosystems. It makes it really easy to be able to pick up and put down another code base without having to slog through whatever the hell the other developers formatting style is, because I do that a lot with some languages to where, you know, like you're reading an absolutely incomprehensible Haskell code base, and it does not look like anything you've ever seen before. But yeah, I think they're definitely very good things for consistency. And then all of the checkers are super useful in, you know, finding where things went wrong. A lot of I do wish that some of the checkers were, I guess, more detailed in the reports, because a lot of times it can be kind of hard to nail down like what's going around where specifically. But a lot of the times, I think that's kind of more of a them sided issue than necessarily a Rust thing. And I do wish some of them had better platforms for it, because a lot of them don't work on Windows, which sucks. I'd like to be able to check out like all platforms. But yeah.

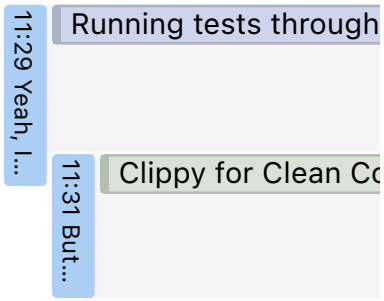
211

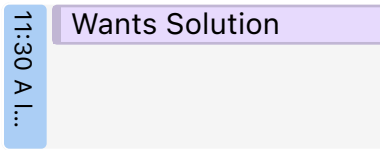
212 **Interviewer**
So working with JIT code, I do have any and with all the FFI stuff that you do. How do you accommodate Miri's lack of support for foreign functions in that context?

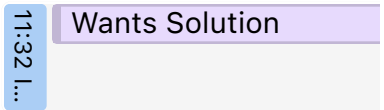
213

214 **Participant**
Um, I mean, generally, you just don't really have a choice. I mean, a lot of times, like, with the JIT stuff, I basically just can't run Miri on them, which sucks. But also, I'm not sure it's really a, I mean, really even possible for









Miri to like, circumvent that without Miri also bundling an x86 interpreter, which would not be fun. For a lot of the FFI stuff, it's not that big of a deal. Or I guess for quote unquote, normal FFI stuff, because generally speaking, you're just writing bindings or something, which don't actually need a whole lot of checking in the first place.

11:... Wants Solution

Interviewer
Is that because the bindings you write don't expose a lot of the sort of object graph of Rust to the foreign code? Or is it another attribute of bindings that makes it such that Miri wouldn't be as useful?

Participant
I mean, generally, bindings are just quote unquote duplicates of what already exists. You know, you have a binding and it takes like a raw pointer to a struct, and then you just immediately dereference the struct and then call the struct's own function. And so like that function would be tested within the course of like normal code. And so Miri covers it, but the binding itself doesn't necessarily do a whole lot that's checkable by Miri, because Miri can't check like, is all C code that could possibly check or possibly call this be well behaved? And the answer is no. So yeah.

Interviewer
Oh, I gotcha. So are you talking about it in terms of Rust functions that are called from foreign contexts and not the other way around?

Participant
I mean, it's pretty much both.

Interviewer
Gotcha.

Participant
I mean, Miri can't verify the correctness of other languages. And so it can't guarantee that the calling or caller or caller context are going to be correct. I mean, it's an opaque boundary that it can't really penetrate.

Interviewer
Gotcha. Have there been situations where Miri, like, where you found some sort of case of undefined behavior across foreign boundaries that like if Miri hypothetically had an interpreter for whatever target that you're targeting, like, Miri could have found it.

Participant
Oh, yeah. I mean, I've written like buggy JIT code that I had to run like, like MSAN and ASAN on in order to figure out what was going wrong where, because you just can't comprehend JIT compiled code.

11:65... ASAN
MSAN

214

233

234 **Interviewer**
So is it generally that the errors that you'd find are things that are covered by the sanitizers anyway? So it's less of an issue that Miri doesn't support any arbitrary like code base that interoperates with Rust?

235

236 **Participant**
Yeah. I mean, like, I'm not sure that I'm not necessarily sure how useful Miri on foreign code is in the first place, because I don't think it's very or I don't think it's really correct or even possible in the first place to apply like Rust aliasing rules to other languages. Because I mean, that quickly becomes like a very difficult thing to navigate of like, oh, this pointer or this like char star pointer is actually, you know, a mutable reference. Have fun with dealing with that. See, like, you know, that's just not like a concept that that language can communicate. And so it's kind of difficult to like, or it's some extent difficult to some extent, unfair to try and machine automate that on there, because you're just going to get a lot of negatives. Negatives is in false negatives where you're missing errors or just negatives is in like challenges or like, well, I mean, Miri's going to scream a lot and be really, really pissed at, you know, like other code, because it's could potentially be behaving wrong or I mean, even is behaving wrong. And those types of errors, like maybe you do have a case where it's like, yeah, this is definitely undefined behavior. But in practice, it's just like not an issue as long as it's passing the sanitizers. Oh, I mean, there's definitely plenty of those. I mean, I hate them, but I think to some extent, it's an unfortunate reality of the world that we live in. Yeah.

11:66 I'm not sure t...
Different FFI Memory I
Miri doesn't s...ort this

11:...
Dynamic Analysis Safe

237

238 **Interviewer**
Gotcha. So like, you have to accept a certain level of undefined behavior?

239

240 **Participant**
Well, no, I mean, it's just more of like, I mean, some other code just doesn't behave well. You know, no matter the language, like some code doesn't behave well. And even though it says it's going to do one thing, sometimes it does another. And so, or you're sorry, could you ask the question again?

241

242 **Interviewer**
Oh, I guess like, is it that there is some like, I guess the way that I'm thinking about this in my head is splitting up undefined behavior into two categories where one category is this is undefined behavior, and I'm observing a segfault, or I'm getting a warning from one of the sanitizers. And then another category of undefined behavior would be what I'm doing is breaking Rust's aliasing model in foreign code. But when I don't get any warnings for this or like, any crashes or segfaults at all from doing it. So it's just going to have to be acceptable because the effort to correct this story, like it just doesn't seem like a reason, like a something that's worth the time to fix. Like, is that dual categorization like a valid thing, or is there a different picture it should be having in my mind?

Participant
Oh, yeah, no, I would definitely disagree with that. For that sort of thing, like, that's definitely not really like acceptable behavior, in my opinion. Like if you're, yeah, yeah, for very like clearly breaking that sort of issue, like, no, that that code is incorrect. And it's, I mean, even though it's not exploding now, it will probably explode later or explode if you turn on like, you know, LTO, and LLVM has access to cross function information.

11:36... Linking Time...timizatio

Interviewer
Gotcha. Gotcha. So it's more that there, there's a certain category of errors you wouldn't be able to find in Miri, you don't expect that those will be a huge issue in foreign code. So it's not something that you're necessarily like super concerned about. But if you did have a magical bug detector that could find all undefined behavior, you'd be fixing all these cases.

Participant
Yeah, yeah.

Interviewer
Gotcha. Okay, gotcha. Gotcha. I think I think I'm following your perspective now. Yeah. So my last question is, are there situations relating to unsafe code in the, I guess in any way that you've used it, or maybe in particular to the JIT stuff, or just your FFI usage, where you feel like a, like the current development tools you have can't solve this problem or this bug, and do you wish there was something that could?

Participant
I mean, I definitely wish that verification to some extent was more fleshed out in Rust. You know, I've like, I've tried a bunch of the tools, like Kani and crucible, crucible, yeah, I think it's crucible. But a lot of like the kind of contract based proofs layered on top of Rust. And they're really cool. But I wish they were more kind of inherently parts of Rust, because, or in like had the ability to express more, I guess, complicated, this like invariants that were required to be able to turn, like kind of hoist a lot of currently unsafe code into safe code.

11:37 I definitely wish... Wants Solution

Crucible

Interviewer
Like that would be. Provide like an example of what one of those invariants would be. Is this something like related to provenance, or is this something that's like more like domain specific invariant that you?

Participant
I mean, probably like mostly provenance related stuff, or yeah, and probably mostly provenance stuff, I guess. Let's see. I mean, honestly, I mean, proofs in general would be a big thing, or it would eliminate a lot of unsafe, because like, I mean, set length on a vector to be able to have

Wants Solution

the constraint that, you know, new length must be less than capacity. And all elements up to new length must be initialized to be able to have those two contracts on that function would be able to make it a safe function. And so, you know, that sort of thing would be amazing, in my opinion. And like, I'm also a big fan of refinement types. I think refinement types would be awesome for us. These are definitely like wishlist dream things. I'm not sure how viable they are, or probably aren't viable, but they would still be amazing for us, in my opinion.

11:38 set... Wants Solution

11:39... Wants Solution

Interviewer
Gotcha. Okay. Yeah. And is there a particular verification tool of the ones that you've tried that you feel like is the best at getting at what you want from it now? Or are they all sort of missing a lot of these features that you feel that you don't, that you'd want in terms of more refinement type style reasoning, or just the types of invariance that you'd want over unsafe code?

Participant
Hmm. I think Prusti is kind of the most complete one that I've used. I mean, it still suffers from the problem, kind of the more or less inherent problem of being an add-on to where there's no guarantee that the code you're calling, or the code you're calling, generally speaking, does not have annotations. And so, you have to either manually add the annotations or treat everything outside of yourself as a black Box, which, you know, isn't being asked for a usability project, because then you end up writing a lot of like template or, you know, template constraints on things. But I mean, that's more or less inherent to the fact that it's an add-on and not inherently part of language.

Prusti

Interviewer
Gotcha, okay. Yeah, sounds good. Well, I think that concludes all the questions that I have for today. Thank you for your time. It's been great talking with you.

Participant
Yeah, yeah.

Interviewer
And then also we're still in recruitment mode. So if you know anyone else who has relevant experience with Unsafe Rust who we benefit from talking to, I'll include a link in that email or you could feel free to just for the, like Reddit link to the study. It'd be great to chat with whoever you feel would be open to it.

256

Participant
Okay, yeah. I mean, I came from the link in, like the dark arts community discord link. I think that's probably a pretty good place to find people if you can get them to come.

Interviewer
Gotcha, yeah. Yeah, we've had a bunch of activity from that particular server so far.

Participant
I think that, yeah. Yeah, a lot of the other people that hang out there are significantly smarter than I am.

Interviewer
Yeah, sure. Well, anyway, thanks a ton for your time. It's been great. I really appreciate it.

Participant
Yeah, yeah. Thank you.

Interviewer
Sounds great. And oh, do you have any other questions I guess for me or any followups about anything?

Participant
Oh, I guess I'm curious about what you all are doing.

Interviewer
Yeah, sure. So my research area is in, I rust broadly, but on one side I study gradual verification and sort of like mixes of static and dynamic type systems for verifying different properties. And the other side I'm interested in sort of usability with rust and in evaluating like a large scale of problems that occur with different code bases. So right now I'm hoping to be able to apply some of the research work we've done in verification to certain problems in rust, but I need to figure out what those problems are. So the point of the study is to add a qualitative counterpoint to a lot of the like, like a lot of the papers on unsafe rust have been about measuring how much it's being used unless like talking with the people who actually use a bunch of unsafe. So I'm sort of trying to provide that additional perspective here, hopefully find some cool niche problem areas where we can use our knowledge of verification to help people.

Participant
Okay, yeah, I guess the end of that interview was definitely speaking to you then. Cause yeah, like, yeah, gradual proofs or a gradual dependent types or refinement types or whatever combination of would be, I mean, that'd be amazing for rust. Yeah, to be able to just prove more non local things. And yeah, yeah, that'd be really cool.

11:68 Ca...
Wants Solution