Advanced Focus Group

KIRAN

[00:00:00] I am now recording in multiple places. Okay, awesome. So, I'm recording on Teams as well. Okay, thank you very much guys. It's very good to have you guys here. So, what I'm working on is a teaching tool for functional programming languages in general, is the thinking, rather than just specifically Haskell, although it is very similar to Haskell, as you'll see. But, yeah, so it's a teaching tool for functional languages in general, and it shows the step-by-step evaluation of this language. It shows the reduction of terms. And I wanted to get your guys' Thoughts on the design of it, the design of the language, and, yeah, stuff like that. So, can we go around? And do you guys want to talk about your experiences with functional languages and functional programming in general, particularly Haskell? Shall we go over and start with you? Yeah, sure.

PERSON 1

[00:01:09] Shall I say my name? Yes, please. I'm [Person 1]. I started programming Haskell when I was in secondary school. I started programming Haskell because it sounded interesting and different from procedural programming, and it was definitely difficult to pick up the language. It was difficult to pick up initially, but a lot of fun, once I did.

PERSON 2

[00:01:37] I'm [Person 2]. My main experience with Haskell specifically comes from the first-year functional programming unit, although I did return in the second year as a teaching assistant for the unit. In the more broad sense, my experience with functional programming, I am working on a dual-context lambda calculus for my dissertation.

PERSON 3

[00:02:03] I am [Person 3]. I became aware of functional programming as a concept during my A-levels when I was like 16 or 17, but I didn't program in it until taking the first year unit about it. At that time, I didn't really get along with it when I did that. I was not very into it, but I have since gained, having done more PL theory stuff, I have become more and more endeared by it over time. When I used it, I didn't like it at first, but I do like it now. That's changed over time.

PERSON 4

[00:02:40] I'm [Person 4]. I was first introduced to Haskell in the first-year functional programming unit. I've never really written anything practical in Haskell, but I've been studying functional languages since then and teaching that first-year unit and types and lambda calculus to third years.

KIRAN

[00:02:59] Thank you guys very much. Just a reminder that it will be anonymized. The names are just to help me refer to you as person one, person two, person three, person four. Cool. Thank you very much, I forgot to open my notes. I'll share on Teams so we can all have it on us. I was hoping to have it on a big screen.

KIRAN

[00:03:56] So the aims of the project are to make it easier to demonstrate functional languages and to provide a platform for messing around and experimenting with them. So this is the platform as it is now. There we go. So for instance, let's just have a look at the UI and I'll just kind of explain it. I will also do some stuff on the UI with my other focus groups. But with you guys, I more wanted to talk about the features rather than the UI specifically. But I do have some things to ask about the UI later. So this, there's a big code window, of course. And then at the top, there's a toggle where you can see the prelude. The prelude is passed before the language, which is why I put it on top. Yes. And there's also, you can load in some example programs. This needs to be improved. I'm working on that. And then there's also a little help menu, which is currently not very good. It will contain the full kind of specification of the language, if you will, and also some kind of information on how to use it. But yeah, so the main thing is this big button here, these two big buttons, where I've got lazy and free choice. So in free choice mode, it provides all of the options for what it sees as the next possible step to make. In this case, these options are, the first option kind of includes the second option. So the first option substitutes the factorial label and then also substitutes five into that. I wanted to, I'll ask more about that later because I need to think of a more elegant way to express that. Let's just, if I pick one of these options, you can see it goes to the next step. So you've got this little thing at the bottom, which is the diff, and it shows what the program currently is under the reductions that have been chosen. So that's how it works. I think it should all be reasonably self-explanatory. I mean, the UI, not so much, but the language, I think you guys should, well, I would, yeah, so if you guys have any questions about what this program does or how it works, and I'll open the prelude as well so you can see that.

PERSON 4

[00:06:55] It's surprising to me that the way if is defined, or then and else. Yeah.

KIRAN

[00:07:02] So then and else in this case are a parser construct. Okay. But that parser construct depends on the prelude being included as well. Oh, I see.

PERSON 4

[00:07:13] So if is a kind of like, parsing-wise, it's a special case, but it's still. Parsing is a special case. It's defined.

KIRAN

[00:07:18] It is still, that is the actual definition of if.

PERSON 1

[00:07:21] So can you generally write functions like that? Because you've got like if, cond, then, else, branch, else, branch. I assume that like those then and else, that is a special case thing, right? Yes. Right, okay. You can't generally just write expressions. It's not high list.

KIRAN

[00:07:37] No. No, that's only for if. Okay. Yes. So I've also got, we'll just stick with the top of the prelude for now. Ignore the data types. Okay. So in, at the moment what I've got here is this whole

kind of section being reduced. It's not really a reduction. It's a series of reductions and substitutions, which I need to think of a more elegant way to kind of express in what way this is a next step.

PERSON 2

[00:08:20] That's more of a de-sugaring, as it were. At least I would say that.

KIRAN

[00:08:22] Well, yeah, I mean it is, I would still call it a substitution because it is.

PERSON 4

[00:08:27] Yeah, I think it still counts as a step too, that you don't replace a name. Yeah.

KIRAN

[00:08:29] I would say it's four steps, replacing the name and substituting one thing, then substituting the next thing, then the third thing. So I've been struggling to think of a better way to represent these kind of multi-step, processes. Do you guys have any thoughts on what might be a nice way to explain that?

PERSON 4

[00:08:57] Um, I think straight away it would be good if, oh wait, sorry, I was just looking into how you could make this UI showing. Yeah. And I guess you could have like, you know when you have the reduction triangle symbol, and then you have like a number or a star next to it, so that there are some number of steps happening, like condensed. You could put a number next to the R and say, you know, this is four steps. And then, like surely, make a drop-down. Yeah. So if someone wants to see what steps are going on inside there, then they could see.

KIRAN

[00:09:34] Yeah. Okay. That's true. I could have a dropdown, I mean.

PERSON 4

[00:09:44] But then I think that would mean, I think I misread the UI a second ago, but I think that would mean that you'd have to sort of, like your program would have to pick which reductions to condense, but maybe you could have some like stock ones that you should always condense, like the four steps involved in reducing if.

KIRAN

[00:10:03] Yes. It would be good to reduce those in every case. So that is a very complex case, but I have thought of a way to get around that, which I'll save till the end, because that relies on the kind of, okay, yeah. I wanted to show you some other programs first before we get on to the next step.

PERSON 3

[00:10:24] Can I just share one? Yeah, absolutely.

KIRAN
[00:10:26] Please do.

PERSON 3
[00:10:28] With sort of, I think one thing that is not immediately clear is how the different reductions you see are related to the main program. If there was some way that like if you hovered over one, you could highlight the portion of the program that it corresponds to. So for example, if like you hide, if you highlight it. Yes.

KIRAN
[00:10:44] I have actually done exactly that. That is done. In a UI that I'll show you later. Aha. So I've got two UIs, one which I'll show you on Figma. Okay. Which does that.

PERSON 4
[00:10:57] Fantastic. I think more originally it would be good if the time were like above the list of reductions so that you could always see it.

KIRAN
[00:11:03] Oh, yeah. Yeah, because it moves around.

KIRAN
[00:11:09] So, do you, just looking at this list, do you guys have any questions about the things it's extracted as next steps?

PERSON 1
[00:11:21] Why is it, yeah, why are there like so many different, because there's, all right, so there's, why, why can we step the if statement but also like, it seems like we can step pretty much any of the statements. It's like assuming that the if statement is false almost here and true at the same time.

KIRAN
[00:11:49] No, no? It's only doing operations on the current state of the program which you can see at the bottom. Right.

PERSON 2
[00:11:58] I am a little bit confused by the, the first two options. Yes. One you've got an if something, then something else something, but then it's also trying, and the second one, it's saying that you can step if to a function that takes three arguments. Yes. So one of them looks very much like a language construct and the other one looks very much just like a naturally defined function.

KIRAN
[00:12:23] So those are both, yeah, so, yeah, so I've got the, those two are the same except I'm substituting in certain arguments. The, so if I do this first one, if I do this one, it will just substitute it in as a big lambda expression, which I agree it would, it is a little confusing that this has the syntax

sugar of if then else, and this doesn't. But these are both valid in the language, so these are both the same language.

PERSON 4

[00:12:57] I think one looks like it's a chain of reductions that contains the other. Yes. The first one is a chain of reductions that contains the second one. I think it's fine to display that as long as you make it visually distinct that these two are related in that way and the other reductions are just independent.

PERSON 1

[00:13:15] If possible, it might be more intuitive and easy to sort through if you could hover over each, or it displayed somehow each section of the expression actually have a thing, because here it's kind of displayed at the linear list where it's kind of nested. There is a nest structure here. There is a nest. So if that nesting could be expressed as you hover over the section and that tells you what the step is there, or there's lines going to each section telling you what the step is, because it's initially a bit confusing and you kind of have to relate it back to what is actually going on.

KIRAN

[00:13:55] That was part of my initial thinking and some of the initial designs I did. Unfortunately, that sounds really hard in terms of UI stuff, which is the stuff I'm least interested in. Right, yeah. But I might just have to do it.

PERSON 3

[00:14:09] I think finding some way to communicate the relations between things.

KIRAN

[00:14:13] Yeah, that's definitely necessary. Should we table this discussion? Because I've got some stuff to show you which I think solves it in some ways, but there'll still be some things left unresolved. I just want to go through the language for a little bit. So let's just have a look at the definitions and things. I guess you guys can go to the website yourself. That might be easier. Yeah, I'm looking at it now. You're on it. Yeah, so you can see the prelude if you just have a look through that. Ignore the welcome message. That's all wrong. That's good. That's from an older system.

PERSON 4

[00:14:59] I hope my laptop isn't blocking that microphone. It's fine. Okay.

KIRAN

[00:15:04] There are many microphones. Yeah, so there's the prelude at the top. And you just want to have a look through that and see if you guys have any questions about that. Or anything you think I should add. I think it should all make sense what all the things do.

PERSON 2

[00:15:34] Do you have. I do. Do you? You can see it in the prelude in the repeat. And take, and some of the other things.

PERSON 1

[00:15:43] Oh, so it's not defined in prelude, it's just the language construct. Yes. Do you have custom operators?

KIRAN

[00:15:50] Uh, kind of. Yes, in some senses, but not in the parser.

PERSON 4

[00:15:56] I think if I was learning special programming for the first time, I would really hope there aren't custom operators. Yeah.

KIRAN

[00:16:02] Yeah, no, I'm just wondering. Yeah, there are, and in the same as Haskell, operators that aren't alphanumeric are treated as infix. Okay. So that's why add and multiply are infix. But the dollar sign isn't in the prelude, and actually it could if I had, it could be if I had custom operators, but I don't. I didn't want to add more parsing complexity for now.

PERSON 4

[00:16:30] Yeah, I think that's good. Not only for you, but also for people learning the language.

KIRAN

[00:16:35] So would you say it would add more complexity to add this kind of custom operators?

PERSON 4

[00:16:42] Yeah, I think it distracts, like, if I'm trying to learn functional programming, I don't think it helps me to be able to define, like, things that have different precedents. I think that, like, distracts from learning how programs are reduced. Sure. And how lazy evaluation effects the ability to do this and the other. Yeah.

KIRAN

[00:16:59] So I've got multiplication, for instance. Do you think it's confusing for that to be infix without being explained?

PERSON 4

[00:17:10] Um, for, like, for just arithmetic stuff, I think that's not confusing, but I would prefer it not to go any further than that. Like, I don't know if you're thinking about it anymore, infix operators.

KIRAN

[00:17:23] No, I've just got, um, comparison and integer operations are the only ones I can think of.

PERSON 3

[00:17:29] I think that's fine, because people are so used to those that it's, like, almost worth it to say anything about it. Yeah. Generally, things that people won't be familiar with can be a fine thing to fix, right? I think it's good that const, for example. Yes.

PERSON 4

[00:17:41] I think it's good that Cons is a prefix, like a normal constructor, and not a colon or something like that.

KIRAN

[00:17:46] Nice. I don't know, just for the sake of, we can move on to a different sample program, if you guys want. If you guys are happy with the preview. Yeah.

PERSON 1

[00:17:57] Interesting, I noticed, I wonder if it would be useful as a learning device to give them a lazy free choice ego. Ah, yes, we'll come onto that later. Okay, right.

KIRAN

[00:18:08] Uh, yes, absolutely. Um, I've got a system in mind where you'll be able to pick the evaluation strategy. Um, if that's what you mean.

PERSON 1

[00:18:18] Yeah. Just so you can see, like, just because I've constructed this thing with infinite from, and, like, obviously, I can go through free choice and actually do the ego evaluation manually, but, like, if I was a learner, it would be cool to be able to see where infinite from breaks when you have infinite, when you have ego evaluation.

KIRAN

[00:18:37] Yes, yes. Um, I will, I'm probably going to add more evaluation strategies and maybe, yeah, more evaluation strategies. I don't think it'll be too difficult, because I've already got, kind of, what I've identified as all of the options. I can probably reduce those down, just to just, yeah. Interestingly, um, in the free choice mode, the laziest one is also always the first one, because of the way I'm, there's the uppermost, leftmost one. It's the, that is just how I'm generating them, which is, I just thought it was cool. So, if we have a look at the, uh, Collatz example, which, uh, because this one actually uses, uh, lists, which is probably one of the main attractions, you know, of this system, in my opinion. Right. So, do you both want to try using this in, uh, in free choice mode, messing around with it as you see fit, and just making sure you, kind of, understand it at all steps, what it's doing? Please feel free to ask any questions. I will just. Okay.

PERSON 4

[00:20:05] I think this, uh, sorry to instantly confuse you about it, but F is now free in this time and I'm not sure where it's coming from.

PERSON 1

[00:20:12] It's from the, um, the, oh, it's just, it's just coming up and, yeah, my bad.

PERSON 4

[00:20:17] Yeah, I should have looked at the source for longer, my bad.

KIRAN

[00:20:21] Does the source make sense?

PERSON 2

[00:21:02] It's possible to construct some humorously large terms, that's been my current strategy.

KIRAN

[00:21:12] You can make it really big and then if you keep pressing the top one it'll get small again because the top one's the laziest, but yeah. I also wanted to discuss at some point we can have a discussion later about how to better break up the big ones, how to display them better-should I have it overflowing or strongly boxed, or I don't know how to deal with that.

PERSON 1

[00:21:45] First of all syntax highlighting would obviously help. Second of all, the backslash con backslash then branch else branch where it shows the compiler's guts and especially where if statements get shown to be match statements-it's very; it's another layer of like it makes the expression bigger and it also makes it very hard to see what's going on because you have to like go through the match statement. Maybe just some like clean up where it's just like 'ah' that match statement like we special case the if which is already special case right and we just don't we don't show it like stepping to a match because if is the fact that it's kind of actually a function is to fill a bit of an implementation detail right like an if statement is generally just built into the language so

PERSON 4

[00:22:38] is definitely like a case of just a language construction I think you can have it both ways in that you can like I was saying earlier like The you could collapse the reduction of if into a lambda and then take the arguments and then it becomes a match statement, you could like collapse that into one thing as long as there's a little line there like icon somewhere where a user can choose to see what's actually going on under the hood and then they can see it in terms of lambdas and matches which are the more core language contracts and then you maintain that you have a small set of core language contracts but you also maintain that ifs don't take four steps to get through yes so unfortunately what I'm thinking of and I'm trying to figure out whether it's necessary

KIRAN

[00:23:26] or not is some sort of structure to say that some sort of structure some sort of some sort of version of the match expression where it's instant where it forces the evaluation of the condition and then it doesn't show up properly and the way I was thinking of doing that is to have it at the top level like in Haskell where you've got so I could redefine it as if true x y equals x and if false x y equals y and then that wouldn't substitute until it was either true or false which would

force the evaluation of the condition that sounds like a pain to implement properly it's such a pain yeah that's an insane level of syntax sugar because

PERSON 2

[00:24:11] That would be three abstractions and then a match expression, I would say stick with the match expressions just because it's very clear that matching has happened when you have the word match there and you don't like also just because it's a bit easier yeah I didn't want it to ever be confusing about why it was doing things I guess

PERSON 4

[00:24:33] I haven't broken it it's just I think something happened when I didn't realise it was happening because I've ended up reducing this at 12 at the same time as the other one and I didn't realise it was happening yes that happens yeah okay I think that's similar to how Haskell does it oh right no I should have noticed this was happening the whole time yeah no it was it's a lazy evaluation with kind of the I forgot what it's called 3.
It's another thing I've heard of I don't know if that's actually the thing but yeah that's it's good that like yeah it's maybe my bad for not noticing it but even if I didn't notice it like early it's still good that this is happening because it's a good thing to start something you can do yeah I guess that like that like makes you internalise what what is it called I've forgotten what it's called I've forgotten what it's called when two times a record come yeah it like you know it syntaxes their equipment then semantically they record them yes and it helps them to return that sound yeah so I've

PERSON 4

[00:24:50] I've done that in an inefficient way but because I'm expecting very small programmes yeah and that could only be like a performance issue right it couldn't be like yeah the inefficiency yeah have a bad experience yeah yeah so what did you guys make of this example yeah it's interesting.

PERSON 1

[00:26:12] I think it's the problem is from my perspective once I have what it's doing it's like, it's kind of once I have the result, it's much easier to see what it's doing but actually stepping through it is almost impossible to follow just because of all of the if statements and stuff like it just makes it, it's like, it's a lot easier for me anyway to just read the code and see what it's doing rather than try and step through it and work out what it's doing.

KIRAN

[00:26:30] Yeah, so the roots of this project are as a demonstration tool right? As a, so the main thing I can think of is manage to crash your entire page yay yeah, I know that there are cases where it'll crash.

PERSON 4

[00:26:50] Yeah I just tried replacing one of the functions with one over zero and then stepped it and oh and then the whole the whole oh is it oh it's stepping by zero yeah okay good I'm fine with it crashing okay the entire thing crashing like yeah what it's rust it panics you but it's like handle

the errors yeah how am I going to handle the errors when they go into javascript well you print a message okay do you want the actual answer you put the yeah I can have a little error there evaluation function in a in a in a catch panel yeah and then you just have you just have to say like oh well I did so I did I did make this UI very quickly yeah it was originally going to be a CLI um I'm glad it's not it it will have.

KIRAN

[00:27:34] a CLI so people can like a full CLI or TUI like um like a ripple you mean no no it's it would be a CLI uh you'd be able to run programs through it um Not compile programmed and file to anything. But because the system is written in Rust, it can have two kind of interfaces. The WebAssembly interface and the CLI interface. So I can very easily make a decent CLI.

PERSON 3

[00:28:12] One thing, this is kind of a UI thing. Because the reduction steps generate bottom-up, it might be good to have some sort of indication about the direction things are going in. Obviously, you can kind of see as they're generated, oh, this is the direction they're going in. But some visual thing to remind you that this is reducing to this, not the other way around.

KIRAN

[00:28:36] Okay, I think you'll really like the new UI then, it does this.

PERSON 3

[00:28:40] It's crazy.

PERSON 4

[00:28:41] I think maybe this is something, having a new one, but it's very, very important that, in the final version of this, if I mouse over this, it will highlight. If I mouse over a redex contraction pair, it will highlight the redex inside the whole program. I would really like to do that.

KIRAN

[00:28:59] The problem is I can't do that by string matching because it might not be correct.

PERSON 1

[00:29:05] Do you store span information? Sorry? Do you store span information when you parse? Because that's the classic way you do this. When you parse, you just store, like in your IR, you just store the span. Each node in your IR just has a span attached. And then when you go and look at this, you're like, oh, I'll just add up the span of this node. No. Right.

KIRAN

[00:29:26] All I have is a line and column. I have the start, but I don't have the end. But I guess now I do because I can stringify it and then add the length. So I guess I do.

PERSON 1

[00:29:39] Yeah. I mean, I wouldn't necessarily store the line and column either. Sorry. I need it

for errors. No, no. But you normally just store the character offset, and then you can resolve the line and column for errors later. Just because if you store just the character offset, you can slice the input buffer very easily. Yeah. I'm not doing that. OK. It's more programs.

KIRAN

[00:30:03] Also, there's no point because if it's passed in as a JavaScript string anyway, it's already in memory. Yes. If I was reading it from a file, I'd probably want to read it in sections.

PERSON 1

[00:30:14] Yeah, but I'm not talking about reading it in sections. I'm just like literally like the classic way is like you store everything with like spans, and then I want to report an error here. I slice the input file, like input buffer or whatever, just here and be like, this bit of text, this is the problem.

SPEAKER_5

[00:30:32] OK. Yeah.

PERSON 1

[00:30:35] But I mean, you've got to.

PERSON 2

[00:30:39] Do you? Are you supposed to allow main to have a type variable? I do. OK. So I've tried typing main with just like A, and then if you just say like main equals 3, then I'll throw an error on the type system. Does it? Yeah. Interesting. I've not seen that. At least on my machine. It's Webassembly. It should be.

PERSON 4

[00:31:03] Because that's saying that main is for all A. Yeah. And 3 is not for all A. Oh, yeah, that's true. It should be an error. Oh. Yeah, that's correct.

KIRAN

[00:31:12] Because that's not true. It isn't for all A and A. Yeah, that would fail. That's just in.

SPEAKER_5

[00:31:17] Oh, OK. Sure. Yeah. Yeah.

KIRAN

[00:31:20] I didn't design the type for example, so. Well, yeah. But you implemented it. No, I did implement it, but I always forget which way around it goes sometimes.

PERSON 1

[00:31:28] Yeah. Another bit of the interface, but you could, the code window could show the error and like underline.

SPEAKER_5
[00:31:37] Yes.

PERSON 1
[00:31:38] I could do that.

SPEAKER_5
[00:31:39] I don't know. I need to remember to. I don't know how I forgot about that.

KIRAN
[00:31:47] That was initially in the plan. I might have time for that.

SPEAKER_5
[00:31:56] Yeah. OK.

KIRAN
[00:31:59] Cool. I mean, that's a lot to think about. That's good. I've been looking for more stuff to do. Can we move on to the, let's have a look at the fix example, just because I think you're guys might like that. It's the same, but I just re-implemented some of the recursive functions using fix.

SPEAKER_5
[00:32:19] So you might be interested.

KIRAN
[00:32:31] This doesn't have any extra language features. I just thought it was cool. Do you think I should put fix in the prelude?

PERSON 2
[00:32:43] I think it's not a reasonable thing to have that. Yeah.

PERSON 4
[00:32:46] I think you shouldn't. I think it's so. Yeah. So I guess.

KIRAN
[00:32:52] I think it would only be for people trying to demonstrate how fix works. Which I guess could be somebody. Yes. Cool. Yeah. It's good to show that. I thought it was cool. It does work. Then there's the filter example, which is another example. I think this is my favorite example, but this example, you guys found the, you've got the dropdown, right? Where the examples are. I need to make that right. But the filter example, I think this is one of the nicest ones because it uses bold. I think pretty cool. Yeah. Filter uses bold. No, it doesn't. No, it doesn't. No. Never mind. No, there's one of the other ones that use bold. But this one's cool as well. But yeah. I want to talk about this example because of what you said earlier, especially in free choice mode. It can get huge with this massively nested match statements and we've already spoken about that. So. Yeah.

SPEAKER_5
[00:34:03] Yeah. Question.

PERSON 1
[00:34:05] Why is this match list explicitly typed as list A here in filter?

KIRAN
[00:34:13] Oh, because I was trying to avoid having top-level type annotations. My solution for that was to do that. But why does it need a type annotation? It doesn't. Okay. It doesn't need it anymore. It is actually unused. Thanks for pointing it out. I forgot. It is valid in the parser. Yeah.

SPEAKER_5
[00:34:33] And it is used.

KIRAN
[00:34:36] No, I guess it is used. It's still used by the type checker. Right. But if it's going to go wrong, it's going to find it anyway. So I guess it's redundant. But yeah. No. I just bit the bullet and decided that I do in fact need top-level type annotations. I haven't managed to find a way around it.

PERSON 4
[00:34:53] I think they're a good thing to have. Yeah.

KIRAN
[00:34:56] Yeah, I could have imprints in some simple cases, but match messes with the imprints. Yeah, Because it relies on unification.

SPEAKER_5
[00:35:06] Yeah, It's not much I can do about that, unfortunately.

KIRAN
[00:35:10] Other than switch to in the . But I don't want to do that.

PERSON 3
[00:35:14] Switch to a different type of . Sorry? Switch to a different type of . Yes.

KIRAN
[00:35:18] I don't want to do that. Yeah. I'll put that in the future work section if anyone could be bothered. But yeah. So the reason I wanted to do this example is because of how huge it gets. Yeah. Yeah. I'm sure you guys have already managed to do that. But I guess I've got to drive it into a ridiculous case where it can-There's like four lines.

PERSON 3
[00:35:36] Oh my God. Yeah.

KIRAN

[00:35:37] It gets worse than that.

PERSON 3

[00:35:38] If you press any button other than the top one, like a lot of times-There's a-I found a strategy which is if it makes-if the thing on the right is bigger than the thing on the left, take that step.

KIRAN

[00:35:49] It's a lot of fun to just destroy it. And thankfully it doesn't hurt the performance too much. Yeah. But anyway, it still works. Even with absurdity, yeah we've already spoken about that in previous examples so there isn't really much more to say about this one. Yes, I want to show you the new UI now, okay?

PERSON 3

[00:36:14] Should we have all the things for that no, yeah.

KIRAN

[00:36:17] Beautiful, not confusing at all.

PERSON 3

[00:36:25] Now construct a type derivation that took my please.

KIRAN

[00:36:28] Yes, yeah. Before that, I wanted to give you guys an opportunity to ask any questions about how anything works, how I don't know why I designed the language the way I had. Any questions at all? Why is if it's like what?

PERSON 1

[00:36:45] Why is it implemented using like both of function and is it literally you didn't want to add like and it's like basically match again into the evaluator so it's just special cases to rewrite that to the tools for the function like what I'm still struggling to understand why, why if is a function and a match expression no, why is the function and not just like, why because.

KIRAN

[00:37:13] I wanted a I wanted to make it clear exactly how it what it was I want to make it as kind of with this few special cases as possible to make it as close as possible to well so my justification as I'm writing it now is I want to make it as close to just simply type down the calculus but why is there a matching question then? Because I need it, I was asked to do that. I think it's good to have that, yeah.

PERSON 3

[00:37:47] So the way I look at it I think matches the like if statements are things that people who

have done any programming will be like, 'I can read that.' But the thing that you want to teach them about is matches and guards, and so having it be that if is like we don't have if it's just a function that uses guards, and then the implementation of if serves as a pretty good explanation of how guards and how matches work.

KIRAN

[00:38:09] But also people who already know that might be annoyed by how massive it makes everything. But then you could just use your own guards so the solution I've come up with for that as I mentioned earlier would be the kind of top-level match expression where it is implicitly a match expression like an asshole where you have a a value that would solve that, but that's really hard, so that's that's why I haven't done that. The other option of making it an inbuilt I had it as An inbuilt originally I it was annoying to because that would add a special case to the reduction and the the, There's the system that finds all the reductions; there are only two things it can find: a substitution or an application. And adding a third thing made it more complicated, but also I just wanted to add as little as possible to that system. But I think the best solution and compromise I think is the top-level match expression thingy. The implicitness of the system I think the implicit match expression would solve that. Do you think it would be better to have it as part of like some sort of mysterious inbuilt?

PERSON 1

[00:39:36] I mean, I don't know; like, I think it just feels weird to me doing it this way, but that might just be because normally it wouldn't. But the if wouldn't be in prelude right; it would literally just be inside the compiler there was a hard code to rewrite that. I really wanted if to be in the prelude; I don't know why.

PERSON 3

[00:39:53] Yeah, I think that's just I think being able to see it go and look at it and see what if is doing I think it's handy for learning. It means that there's a sort of a route an easy route that if you're using this to try and understand how a functional program works, you can think I will think of a clause as an if statement and then sort of translate. That through the 'if'

PERSON 1

[00:40:17] function, I do want to point out that exceptions are bad for learning so the fact that like you've got 'if then else' and 'if' is the only function in the universe that has this 'then else' I don't know if that's a good thing I don't understand why you did it because like 'if'

PERSON 2

[00:40:28] with three arguments where the first one is in prelude yeah it just means a lot of brackets the thing about it that was confusing to me is the fact that it is both part of like a sugar and it's also just a function in the prelude you have this like dual meaning of 'if'

SPEAKER_5

[00:40:43] I see that yeah yeah yeah whereas

PERSON 1

[00:40:48] 'if' it was just sugar right and the compiler like inside

PERSON 3

[00:40:51] Itself rewrote it to a match that'd be very, very, very tough or it was just interesting. I like that actually I like that I don't get it; it just means sugar because then you still would through evaluating you would still see that it is how it works as a match but you wouldn't have this yeah double meaning.

PERSON 2

[00:41:13] I mean you could use like even to me like a super simple fix would just be in the prelude rename if to if else like if underscore else because then there is no longer a double meaning between if and if yeah and if the function yeah because you're not you're not shattering the function name yeah right but they aren't two different

PERSON 1

[00:41:33] Things I guess, but it's confusing, it's an internal thing because one's a parser construct and one's a yeah, but they've got the same name, it's defined as a function and can't pull functions with if then else right, but that's a five-argument function normally, yeah yeah.

PERSON 4

[00:41:51] I think I need syntax, I think at the least [Person 2] what you're suggesting is like the same thing that could be achieved just by having the sugar like we're talking about like if the if construct was fully an inbuilt thing and there was no entity in the prelude at all but it was reduced in one step to a match statement yeah then you could still say the meaning I think. There's many solutions, I think it would be better to not clutter up the prelude with something that kind of already exists, yeah.

PERSON 2

[00:42:25] The issue I was having is just the fact that there is a function in the prelude which has the same name as some syntactic sugar that is a parser construct and like I don't think having two things with the same name that are fundamentally different parts of you know completely different one's a parser construct one's a function, that's defined. Having that mismatch is just a bit yeah skewed in my mind, yeah.

SPEAKER_5

[00:42:54] Yeah, okay.

KIRAN

[00:42:56] Yeah, I'll have to think about how to resolve that. I think my option with the top.

PERSON 4

[00:43:04] level match expressions is a good option for that as well but and then I'd also have to remove it from the prelude or make it clear that it's not the same thing yeah very good point sorry

I think top level match expressions wouldn't be that good because it's like I think in themselves they are kind of a special case like as a Haskell programmer I'm sure everyone appreciates them because they look good but if you're just learning if you're literally new to the reduction of terms then it's not obvious that this is kind of sugar for a term like something on the left side of the equal sign of a function definition it's sugar for

KIRAN

[00:43:43] Something on the right side, I mean I've already got that with the abstractions so this is sugar for abstractions I suppose but it would be good to keep that yeah okay okay I'll have to think about how to disambiguate if in the prelude and yeah yeah thank you very much I think the sugar is the best option wherever it is yeah like a special case for this match expression is an if treat it as this don't show it in the reduction I guess that would work fine yeah cool I'll have a play around with that see what see if I can make that work well nice thank you very much yes I want to show you the Figma thingy I'm not sure if I can share it unless You guys already have Figma accounts, so I will just do it on my screen.

SPEAKER_5

[00:44:44] Cool, okay!

KIRAN

[00:44:55] I'll just do it on my screen, so in this version, it's pretty similar except the 'oh' is it still on the other yeah, it's not showing the 'oh'. I'm doing all the

PERSON 2

[00:45:07] 'oh's, Teams done the Teams thing where it just stops like updating the screen.

KIRAN

[00:45:10] It was I said it was I was sharing the whole screen um, I've had a few issues with Teams before.

PERSON 2

[00:45:18] It just selects a frame and decides that this is the only frame ever, like what?

PERSON 3

[00:45:26] So if I do what, a wonderful piece of software, thank you! I think you should click the entire screen.

PERSON 4

[00:45:31] I don't think you did anything wrong, it's just a bug, we're

KIRAN

[00:45:33] At the video stop, is it oh it's still showing oh there oh my god wow that's slow is this yeah that's quite a while ago that is diabolical, okay well thank you MichaelSoft MichaelSoft, you have to count the frames per second on hand two, I think that's enough um yeah, so the prelude

is here now uh the in terms of the input box I moved it to the left because it doesn't need to be so wide yep, unfortunately because of the way I'm doing that input box I can't make it like a dynamic size, I would like it to be able to be resizable but the library I'm using doesn't allow for that, can you do like a scroll to zoom the text in and out at least no no okay. So, it's a fixed-size yeah, unfortunately unless I switch to a new one which I might, um, we I mean I guess we you just changed the size of it, you can't change it, you just did what well vertically yes yeah yeah yeah so I can change the size but it's not sorry is this just because the library doesn't, yes, the library doesn't I'm going to try to find a way around it um I don't know I need to learn how to do React properly but uh it doesn't have that functionality out of the box to have like a uh uh dynamic size it you set it when the component updates um okay yeah anyway so you've got um I added this evaluate to completion mm-hmm I thought that Was useful, mm-hmm, because sometimes you don't want to useful to see the results and also, I added does that show you every step along the way or does it just show it? It just shows you the results so it says, 'Yeah

PERSON 3

[00:47:13] ', so it's like, what does it this evaluate to this? Yes, if you're interested, yes, yeah.

KIRAN

[00:47:18] I think that's it, that's it with the steps because I thought it would be nice if this was used as a lecture tool, yeah, which I hope it would be. Uh, that allows you to reference like specific lines, mm-hmm, more easily, yes, and then there's also an ordering, yeah, and it shows the ordering, like I'm saying earlier, yeah, so then I'll have like a little drop Down with evaluations which is cool but yeah Yeah, so the thinking I've been, that I've had an idea of, yes, as you guys said earlier, this is not just a list of things to do. These have a structure to them. And here I've kind of tried to say that by having this, it just says 'do below at the moment', where it says 'do the one below it' and then also 'do this'. Do the one below it and also do this. I think I need to think of a more elegant way to say that. Maybe I just need to remove the words 'do below' and just put it in the help menu and say it always does the below.

PERSON 4

[00:48:29] The only way I can think of is you can reverse the order and then say on each of the ones that are not the first ones, say dot dot dot and.

KIRAN

[00:48:36] Yes.

PERSON 4

[00:48:37] I think that would work pretty easily, but I don't know if reversing the order is a bad idea.

KIRAN

[00:48:41] Yeah, but then that wouldn't work so well with the free choice mode where they're not all related to each other. I need to. I need to think of some way to connect them.

PERSON 2

[00:48:49] Is there some way such that like if you hover a mouse over one of them, it'll like highlight the other ones that it will also do or something? Oh, yeah. Yeah, yeah, yeah.

KIRAN

[00:48:59] I like that. It would highlight all the ones that pressing that button would do. Yeah. Yeah. Yeah, that would be good. Yeah. I've also removed physically on the button what would be the results because it's not necessary, I don't think. That's true.

PERSON 3

[00:49:17] It also, there's a little interaction there of like testing if you understand what that means, if you're like, yeah, yeah, I'm going to, I think this will do this, I'm going to imagine what it's doing, then click it and then I see, oh, yeah, it does do that.

PERSON 4

[00:49:30] Yeah, it allows it to do the priority battery.

PERSON 3

[00:49:32] Rather than it just like, the answer is in front of you. Yeah, that's good. Or when the answer is being put in the direction. Yeah, cool.

KIRAN

[00:49:38] Yeah, that works. That works very well. So, yeah, that's a nice justification for it. That's all right. Thank you. Thank you. Thank you. If I click on, this is an interactive thing, which is cool. It took ages to make in Figma. So if I have, if I substitute filter, for example, it will do this. And I've got this here where this kind of emulates the fact that if I hover over the apply selected attraction to is even, it will highlight, as you say, the bit of the text that that would affect. Yeah. I didn't want to regrade this in Figma. Yeah. Also, Figma doesn't have highlighting. That's so strange. You have to, I had a little yellow box behind everything, so I underlined it instead of highlighting it, but I'll highlight it probably, and it'll be different depending on what, which one you select. Yeah. Yeah. So I very much agree with what you guys were saying earlier that that would help. But yeah, so you have these options which are kind of inclusive of each other. I think I want to think of maybe a better way to show that. Especially in the kind of free choice mode, which I'll still have, but lazy will be the default, but I have like a free, but I've got lazy twice, but if I have like a free, free choice mode, then yes, that's a really good idea of having it highlight what it would also do. I think that solves that problem. Yeah. I can also have the MIDI be in big. That's a good idea. Yeah.

PERSON 3

[00:51:12] Having like a few. Yeah. There's solutions. There's a thing though. Like eventually that doesn't work forever because you run out of space to invent. Yeah. In time. Hopefully you're not displaying that many items on the.

KIRAN

[00:51:24] But maybe I just don't care about that because this is.

PERSON 3

[00:51:29] Or maybe it only indents when you highlight, as well as highlighting it indents them at the same time.

PERSON 4

[00:51:34] I think you can just indent all of the non top level ones, right? Yeah. So the top level ones are.

KIRAN

[00:51:43] It's difficult to. To really explain what they are because they are normally a substitution and then a series of applications, and it's hard for me to express that these aren't kind of reduction steps. Yeah. These are transformations by another definition. Yeah. Yeah. So I've got hopefully all that information will be in these, in these help menus. I'll have a CLI available to download, it's like a binary probably. Yeah. So do you guys think that this is stronger than the paraphore? Definitely. Yeah.

PERSON 3

[00:52:30] I think having the vertical split instead of the horizontal split.

PERSON 1

[00:52:36] Yeah. Much better.

PERSON 3

[00:52:37] It's easier to have everything on screen and it's more akin to what people may have experienced. Like. Like other online language tools, it's nice. Like, it's not common to have code on the left and compiled or like result on the right, like Compiler Explorer. Yeah. Compiler Explorer.

PERSON 4

[00:52:54] Rust playground.

PERSON 2

[00:52:56] Yeah. Like any, anything like. Yeah.

PERSON 4

[00:52:58] I think immediately not having to scroll is a massive plus. Yeah. Huge. On this one, I have to scroll all the time. Yeah.

KIRAN

[00:53:03] So I want to make it so you don't have to scroll. Does the text wrap at the moment on the left box? On the left. No. But it should. That's a good point. Okay. Does it scroll if you don't wrap? It does. It horizontally scrolls. Right. Right. Yeah.

PERSON 1

[00:53:19] So it's disappeared off the side of the screen. I think I'd rather have it wrapped because I've got line notes. I disagree. I, I, I'm personally, I hate, a hater of, of soft wrap. Cause you can put a new line in there yourself. Exactly right. Like I'm not going to break the pattern.

PERSON 4

[00:53:35] Is that true? Can you put a new line in, like in the hybrid and expression? Is that going to help? Okay. A new line.

PERSON 2

[00:53:41] It's the end of an expression. Would it, would it be controversial to select a button that says enable soft wrap?

KIRAN

[00:53:48] I don't want to do that. Okay. Um, I can have a whole bunch of editor options and stuff, but I don't really want to, cause I don't anticipate people ever using the editor apart from like a vague amount of messing around, but nothing serious.

PERSON 1

[00:54:03] I don't like soft wrap because it had a tendency to make it very confusing for code to follow in my opinion. But, um, But I put the line numbers.

KIRAN

[00:54:11] Wouldn't that fix that?

PERSON 1

[00:54:12] Well, no, because you still have to like, it's like, oh, sometimes the line number, like a new line is not actually a new line. They have to like jump. I'm like, oh, especially if an expression is terminated by a new line. Yeah. I'm like, suddenly I can't just look line by line. I have to be like, oh, this is actually part of the previous expression.

PERSON 2

[00:54:26] We need to get this user on Helix right away. What? I mean, just Helix has a really good job of showing when I'm wrapping, when a line is wrapping in my opinion.

PERSON 1

[00:54:34] I have dyslexia. Go find yourself. Oh, okay. Yeah. That's my like, ableism card.

KIRAN

[00:54:40] Well, cool. Um, another thing I thought I'd add is the ability to step back by clicking on a previous table option. Yeah. If I go to the front, I can step back to this step. Oh my God. Yeah. That's so good. Which I think is nice. Beautiful.

PERSON 3

[00:54:59] Something I had not thought of, very good. Sorry? Something I had not thought of,

very good. I think generally it feels a lot more structured now. Yeah. Whereas it kind of felt like, 'oh, I'm just getting lines and lines and lines and lines.' And like once they're done, when they're done, they don't like, it's kind of hard to look back at them and understand them. Yeah.

KIRAN

[00:55:16] It was initially very slapdash. That wasn't a design. That was just, that was a proof of concept. Yeah. I guess. Okay. No, I'm glad, I'm glad you like this more. Do you have any, uh, other things you want to discuss about this or any suggestions?

PERSON 2

[00:55:35] Um, not in particular.

PERSON 1

[00:55:37] Wow. This is pretty cool. Uh, I mean, I'm still. I'm still partial to the whole, like, obviously highlighting is a bit difficult, but like, like if when you hovered over the, actually, I kind of already, no, no, no, it kind of, yeah, no, it does do the substitution there.

KIRAN

[00:55:54] It will, um, I'm not, I'm hoping either I will, I will remove free choice mode and have it. So you just pick an evaluation strategy in which case all of the things will be related to each other. So I will just be able to highlight the same bit. Or I figure out a way for, to add in the highlighting. It might have to just be a string match.

PERSON 1

[00:56:18] I do think free choice mode is useful as like a teaching tool. Probably not that much. Maybe it's useful for someone who's like a self-learning tool, but I think as like a teaching tool to be like, 'look, we could look at this expression.

PERSON 4

[00:56:30] It's fun to make it really big.' I think it's worth, um, doing highlighting the proper way and I think, um, maybe it's scary, but I don't think it's going to be as hard as it sounds.

KIRAN

[00:56:40] I would have to restructure a lot, especially, the main problem is the fact that positions change when you reduce things.

PERSON 4

[00:56:51] Well, so do the, so do the options, right? You're going to have to be updating the buttons and what they do anyway. No, that's, that's different.

KIRAN

[00:56:59] How am I going to be able to highlight it in the kind of source text?

PERSON 4

[00:57:03] Oh, are you thinking of highlighting over here? Well, yeah. Because I was thinking of highlighting within, within the term that's just below the buttons. Which. Which sub-term is actually going to be reduced by clicking on that button. Yeah, I don't mind about highlighting in there. That's not important to me. It's, it's just in here.

KIRAN

[00:57:15] So in my mind that would require, so this is not, so it, the way it works is it doesn't spit it out and then repass it. It maintains the tree structure and modifies it. If I wanted to do it like that, where I have, um, highlighted bits kind of moving around. I'm not, I'm not sure how I'd do that. Uh, I think I'd have to restructure how the, maybe I'd have to have a whole extra system that keeps track of bits. And updates things with where they are now in the source text, now that the source text has changed. Um.

PERSON 4

[00:57:58] I don't, all, all I'm hoping for is like a map brain. Um. From each button to the, to the span in, in this string specifically of which sub-term it is. Yes. That should be fine. And so I think you can, you don't have to like keep track of anything across reductions. Surely. Because there is some, there is some syntax tree inside your program. Yeah. And then, while you're formatting it, um, you would be able to say, okay, this is where this sub-term is starting, this is where it's ending. Even if you just put like three capital X's followed by a word which represents a color and then like grip that out and replace it with some HTML. Yeah. Um.

KIRAN

[00:58:43] That is kind of how the diff works. Nice. Um.

SPEAKER_6

[00:58:47] Yeah.

KIRAN

[00:58:48] Okay. That makes sense.

PERSON 4

[00:58:51] I'll have to think about how to do highlighting. Um. I think it would be so, so helpful. Like I'm sorry to, to go on about it. But I think it would make such a big difference. No, no. I think you're right.

KIRAN

[00:58:59] I just didn't, I just don't want to. I probably should though. Yeah. Um. I'm not sure if I've got time for it. Yeah. Um. I'll have to think about it. Yeah. Yeah. Um. Well, I think that's, um, that's very helpful for me to know probably what I should do even though I don't want to. Because I've been avoiding thinking about it.

PERSON 3

[00:59:19] Yeah. I think, unfortunately, as much as it is not the interesting part of the project, the

UX. Well, that's all done.

KIRAN
[00:59:29] Yeah.

PERSON 3
[00:59:29] The UX is kind of like, for it to be effective at teaching. Well, exactly. Hence. Hence this. Yes. Hence everything. Yes.

KIRAN
[00:59:38] And I think I'd need a full supervisor. Ooh.

SPEAKER_5
[00:59:40] Is that right?

KIRAN
[00:59:41] For this exact reason. Yeah. Because I knew in order for it to be like a good teaching tool, I'd need a, I'd need a not, a not PL person.

PERSON 1
[00:59:51] That's real. Have you ever tried to use any of the languages designed by, like, pure PL people? Yeah, well. They're horrific.

KIRAN
[00:59:57] No, I haven't, and I don't want to. Yeah. Good. That still is an anomaly because it's feasible. Yes. Yeah. Um. So I'm hoping my, whoever's my second marker will appreciate the utility of this.

PERSON 3
[01:00:09] I think this is very good. Yeah. Okay. I always hated the parts of the functional labs where I had to leave it. Like, it was like, to show you, to show you understand evaluation, evaluate it by hand. The amount of times I've done that. This is annoying. I don't want to do it. Yeah. And then I wouldn't do it. I wish I'd had this done by the functional labs.

PERSON 4
[01:00:27] That would be, I hope they use it next year.

PERSON 3
[01:00:29] Yeah, that would be really cool.

PERSON 4
[01:00:31] And I hope they keep in touch with you, and they do. Yeah, that'd be cool. So they can say, 'Look, it's going really well.' Nice.

KIRAN

[01:00:36] Get, get the occasional bug report. Yeah. Yeah. Yeah.

PERSON 1

[01:00:42] That's something I wanted to talk about. I almost asked that question.

PERSON 3

[01:00:45] It was like, that seems like a bit too far. Yeah, that's what I thought as well.

PERSON 1

[01:00:50] Because it would be, I feel like, incredibly useful to be able to step through what a Monad is doing and be like, 'Guys, F map.' I don't need type classes for that. No, no.

KIRAN

[01:01:00] I can, I can, so I can implement, have, I can have like a special case for monads and have it. See if I have like, I don't know. a monad with a say if i had over over strings with like input and output for instance if i did uh i can implement i can write or bind i can write bind in the language but i just have to do some special cases for a the types i'd have to add some sort of like real world or something something that represents that in the type system which i can do i had the thought i have i have done it but it just doesn't do anything i think it's a bit much i had the thought i wonder if there's type classes i thought type classes kind of imply monads that's too far yeah like it's like if you if you have type classes you would want to have monads because that's like a good example but like once you have type classes then everything is like like they're not i don't think i'd need type classes i just you don't have to write a special you just need to write like a special case for each type each monad i also think that type classes are a static thing the whole point of understanding monad i would hope you understand how tongue evaluation works yeah like i feel like if i evaluate how a functional programming language evaluates is that is a step on the path to understanding functional programming and monads is discreetly later

PERSON 3

[01:02:20] And like I don't think the best way to understand monads is to see how they evaluate, yeah.

KIRAN

[01:02:25] So then, so the way I've been kind of going about deciding what features to include and not include, I've been looking at some university courses and starting from the beginning, yeah, and seeing how far they're going to go and what features they're not going to include, and how far I could push this system through, yeah, and without it becoming ridiculous. And I think that's where it stops because most of the courses are taught with type classes and stuff at the end, so pushing it, I think the system has designed in terms of language allows.

PERSON 3

[01:02:57] Me too, push through as far as possible as most universities courses of as many university courses as I can, yeah, without it not being a bachelor's project anymore, yeah, um, I

think beyond that, I don't know, I think it wouldn't be too hard to implement type classes but I think it'd be unnecessary, yeah, I agree came to conclusion, I don't think it was necessary, I think yes the set of features this has a really good set of features, I think okay cool, the way list is defined was great, I think either even maybe list is an iconic choice either maybe a list, yeah, iconic choice of structures to have that's all over the big three. Technically speaking.

PERSON 1

[01:03:38] You don't even need maybe, like you've got either and you have this, like you're done.

KIRAN

[01:03:42] Yeah, I guess if I had unit, I could have, I don't have unit though. Oh, do you not have unit?

PERSON 1

[01:03:47] How have you got an FP language without unit, what the fuck? I didn't need it. Yeah, data unit, right, but like.

KIRAN

[01:03:54] I didn't, why would I need it? I know, in all the algorithms they had unit but I couldn't think of what I needed. But the category theory.

PERSON 1

[01:04:02] Yeah, it's the fucking unit type, it's the term, the initial object of everything.

KIRAN

[01:04:08] Initial in what way? Let's not go there. Yeah. Category theory. I think I might be beyond me. Category theory. You can define unit if you want.

PERSON 4

[01:04:18] Wait, so it's not the initial object, it's the terminal object. Fool. I'm going to stick to what I said earlier, it's good that this programme shows you about dynamics and you should avoid trying to show anything about statics. Yeah. Like there's no point in improving the statics capabilities of the language because for what reason? Yeah.

KIRAN

[01:04:33] I could add unit to the parade, I guess. I could have data unit equals unit. That's true.

PERSON 3

[01:04:39] I would say, as far as like the university's unit structures, statics, you get statics in year three. Like you don't get, in the PL units you don't really get much about statics in programmes until.

KIRAN

[01:04:55] What do you mean by statics, sorry?

PERSON 3

[01:04:57] As in like, dynamics being how programmes evaluate. Yes. How programmes are typed. Oh right, yeah. At the theory level. And again, you are told what a type is, but you are not told about, you do not learn about type systems or anything related to them until third year. Yes. Whereas you do dynamics in first year and second year. Yeah.

KIRAN

[01:05:15] So this is aiming first and second year. Which is quite dynamic. Yes. Given that I've only just learned. As a third year myself. Yes. I think I didn't really fancy going any further into the types than I did. I just found an off-the-shelf algorithm and implemented it. Do you think there's anything else needed in the type system? That was a question I was going to ask, I guess. Probably not type classes. I think there's nothing else.

PERSON 3

[01:05:41] I think, keep it simple.

PERSON 1

[01:05:44] Kiss. Kiss it. Yeah.

PERSON 3

[01:05:47] I think it's like, because as, I think, did you say your goal was to keep it as close? No, that would be the thing. I don't remember. Did you say your goal was to keep it closer to like simply type lambda calculus?

KIRAN

[01:05:59] Pretty much. Yeah. With as few changes. With name functions, I guess. With name functions and match statements. Yeah. And like tag unions and type aliases. Yeah.

PERSON 2

[01:06:10] Remind me, do you have any product types? Oh, you have constructors with-I also, yeah, but I also have pairs explicitly.

KIRAN

[01:06:18] You have pairs. Okay. I've got N-tuples. N-tuples. Yeah. So I have all algebraic data types. Yippee.

SPEAKER_6

[01:06:24] Yeah.

KIRAN

[01:06:25] I guess I don't need N-tuples, but I have them. Yeah. If you've got like N-tuples.

PERSON 2

[01:06:30] If you've got like a data A, B, C, then you've already got like N-tuples. Yeah. Just tag them explicitly. Yeah. Yeah.

KIRAN

[01:06:36] But no, they're parser inbuilt. That's sort of nice. And I added them before I added tag unions. I thought it would be easier. Yeah.

SPEAKER_5

[01:06:49] Yeah.

KIRAN

[01:06:50] Okay. Cool. I wanted to use whatever time we have left. What's the time? To see if you guys can break it. Oh, yes. Yeah. So we can write like a division by zero. Division by zero, you're right. I need to add stop crashing it.

SPEAKER_5

[01:07:10] I don't know.

KIRAN

[01:07:10] I decided that wasn't important, the fact that it crashes. I thought it represented something nice, but there are some expressions that do crash it. And I just found one this morning. But the one I wanted to see if you could break it, the thing I wanted to see if you could break was the type checker, if possible. Oh, God. Right. Do you have any guidance on how we break the type checker? I haven't found a way. I had a big list of bugs, but I think they're all resolved now.

PERSON 3

[01:07:46] There's no let, is there?

KIRAN

[01:07:48] No, but you can implement it using either a match statement or an application. I will add let as syntax sugar.

PERSON 3

[01:08:03] I'm trying to think of difficult edge cases in typing letters.

KIRAN

[01:08:14] I could add a file full of untypeable things. The file of untypeable things? Oh yeah, that's all out of date.

PERSON 4

[01:08:25] Oh good. What are the subtyping relations? Because this error message is talking about subtyping, but I don't know.

PERSON 1

[01:08:34] Oh yeah, you could make the error messages. The error messages need improvement. I cannot figure out how. Maybe either A int could be subtyped, or maybe either int B. Subtype failure.

KIRAN

[01:08:42] Yeah, so the subtyping doesn't have any mechanism of producing error messages. So that's why it just says subtype failure. And then whatever step was above that tries to reassemble that into a default. That's a very recent error, but it fails. Subtyping, I meant to print it out, but they seem to be charging us to print again. Oh good. What? I meant to print it out, but I didn't have 60p on my print credit. No, mine was fine. A UTA? Yeah. Huh. I printed something out quite recently, but it tried to charge me this time. It just failed.

PERSON 3

[01:09:18] Which print did you use?

KIRAN

[01:09:20] I tried the one in the school office, and I tried the one upstairs. I tried throwing both ones upstairs.

PERSON 3

[01:09:25] I would just go to the ass for printing, because I trust it. I love printing in the ass. There's computers there, so you don't have to deal with the email interface, so you can just print the fucking thing. Yeah.

KIRAN

[01:09:39] Yeah, so finding bugs in this is really hard, in the type checker. I can't prove that there aren't any.

PERSON 3

[01:09:51] I guess you can't.

KIRAN

[01:09:52] No, I can't prove there's no bugs. I can prove that the algorithm as designed works, but I can't prove that they implemented it correctly.

PERSON 2

[01:09:58] I'm trying to throw edge cases at it, like defining other types with the same constructors as existing ones, and you seem to have covered those cases. I think I've covered those cases.

KIRAN

[01:10:06] That should all be caught in the parser. The parser keeps a track of everything that's bound at every point.

PERSON 4

[01:10:14] So what is a subtype of what, except types of themselves?

KIRAN

[01:10:19] So, the literals, integers and digits. The booleans are obviously not subtypes of each other, but they're subtypes of themselves. If I have, oh, you won't like this, but if I have like a tag union data type, the way it defines subtypes is string matches the name, and then does the subtype over all of the things in it. I don't like the fact that it uses a string match, but I think it's correct. Given that the parser enforces that there aren't two with the same name. Also, there's no matching identifiers, so you just have to do a string match. Well, for the type itself.

PERSON 3

[01:11:03] I've encountered a kind of strange thing with the parser, which might be correct, and I'm just being silly. But lambda x dot x without a space is fine, but then you throw a 2 in, and I assume it's because it's trying to read it as a 0. 2.

KIRAN

[01:11:17] Yes, that is what's happening. That's bad.

PERSON 3

[01:11:22] I don't know if that's like, it should be like that.

KIRAN

[01:11:24] That's a problem with Alexa. God, that's really stupid.

PERSON 2

[01:11:30] Are you on your own Alexa, or are you using like logos?

KIRAN

[01:11:33] No, I'm doing everything myself.

PERSON 2

[01:11:34] I wondered if it was like, it was trying to parse it with another one.

KIRAN

[01:11:37] Yeah, the only library I'm using is WASM bind 10.

PERSON 3

[01:11:42] Because if I throw parentheses around it, it figures it out. Yes.

SPEAKER_5

[01:11:50] Yeah.

PERSON 1

[01:11:50] Doing something. These kind of edge cases, I've recently discovered an edge case like this in the Rust compiler.

PERSON 4

[01:11:58] Are you saying this program should what? Like, should A1 be a subtype of A2?

KIRAN

[01:12:05] No.

PERSON 4

[01:12:06] Okay. Because I don't know what is a subtype of what.

KIRAN

[01:12:10] Other than A. Yeah, that's a good point. I need to write a lot more about the typing rules in the, so no, they should not.

PERSON 4

[01:12:18] Because so far it seems like nothing is a subtype of A.

KIRAN

[01:12:22] Unless it's self, pretty much. So subtyping, it's probably more of an internal thing. And maybe I should keep it that way without putting it in the error messages. So subtyping is just when I have what is in this algorithm referred to as an existential, which you probably think of as more of a unification variable. Yeah. If I have an existential, if I have a unification variable, the subtyping algorithm will check whether what it currently thinks it is, is a subtype of the new bit of information it's got.

PERSON 4

[01:12:59] So if it's polymorphic. You could check that list int is a subtype of list A maybe. Yeah. Right.

KIRAN

[01:13:04] That's what it is for.

PERSON 3

[01:13:06] I would say A1 is a subtype of A2 by like pure definition of subtypes in my head. Yeah, I don't like that.

PERSON 4

[01:13:19] There's no such thing as that. A2 can be more things. But also the parser should prevent this. If we consider these to be A1 double column A and A1 double column B, and this is A2 double column A, for example. This is actually wrong.

PERSON 3

[01:13:35] I see that. I see that. Yeah. I don't think you're going to run out of pure meaning.

SPEAKER_5
[01:13:43] Yeah.

KIRAN
[01:13:44] The parser should stop that. Okay. The same name. Because these A and B should be bound at line two. Yeah. Those are. Yeah. I'm treating them as functions. Constructors in my system are functions that don't evaluate. Which I think is fine. It makes sense because they don't reply to things. Yeah. They're just functions but with capital letter names. That is the only difference. And any function with a capital letter name does not evaluate. Yeah. I mean, it's not the whole problem. Yeah. Exactly.

PERSON 3
[01:14:21] Yeah. We figured it out, guys.

KIRAN
[01:14:25] I mean, that's the example. Oh, and also another thing I want to ask. Do you think I should add an untyped mode? I can just turn off the type checker. Oh. So for instance, if someone wanted to use the Y combinator or like. Well, what? The Y combinator is fixed, isn't it?

PERSON 3
[01:14:43] No.

PERSON 4
[01:14:45] The Y combinator is a fixed point combinator. The Y combinator is a fixed point combinator. But it is not fixed. Because you can't type the Y combinator. There are also other fixed point combinators.

PERSON 3
[01:14:54] There's like a, the Y combinator is untyped.

KIRAN
[01:14:58] What is the Y combinator then? Oh, it's.

PERSON 3
[01:15:01] Lambda F dot lambda X. Lambda F dot, yeah.

PERSON 4
[01:15:03] There's a couple of X dot X Xs in there. Yeah.

PERSON 3
[01:15:05] Lambda F dot X X dot lambda dot X Xs. Yeah. It's that. That's the one. I'm familiar. It does it twice. It kind of.

KIRAN

[01:15:12] It does it twice. Yeah. For instance, the classic untypeable thing, I guess, lambda X dot X X. Would you ever want that? Is it worth adding an untypeable mode?

PERSON 4

[01:15:27] I think you should add this because the only term we've come up with that you can't type that you might want to use is the Y combinator, which is introduced in third year. So just by the way, this is aimed at first and second years. We don't need that. But it's easy to turn off, I guess. That's also true.

PERSON 3

[01:15:42] I actually hate Y combinator for the company for making it difficult to Google what the Y combinator is.

KIRAN

[01:15:47] It's tattooed on someone's arm.

PERSON 3

[01:15:49] Yeah. This is it tattooed on someone's arm, which is lambda F dot lambda X dot F X X. Yeah. But anyway, it's untypeable. Lambda X dot F X X. It's untypeable. And fix is just like we say, I'll have the chat fixed. Fix. Yes.

KIRAN

[01:16:02] But I don't even need that because it just has, I think, name replacement. Name. Yeah, name replacement. Whatever you call it.

PERSON 3

[01:16:09] I think having name functions in there. Yes.

KIRAN

[01:16:15] But then fix can allow me to do recursion over like something I don't want to label. Yeah. If I don't want to label something.

PERSON 3

[01:16:23] But you can implement that with your name. Yes. With your name. Yeah.

KIRAN

[01:16:27] Okay. Cool.

PERSON 3

[01:16:29] Yeah. I. It's good. I think it's cool. There's something else that I didn't, other than that point, do you think in the past that. Yeah.

KIRAN

[01:16:37] That's very valuable though. That's good to know. Yeah. I mean, this will, this will work. This will be available at this address. Forever? Let's get our pages. Because it's static. Yeah. Beautiful. True. WebAssembly is great. This runs on my smartwatch. Whoa. My smartwatch is out of charge. Otherwise I would have shown you. I was thinking you were saying you just. Yeah. Yeah. Because it's not the same. I have a web browser on my smartwatch. Sorry? How do you press the buttons on your smartwatch? It's really hard. And you can't type things. Like. Well, no. And you can't exit the thing once you've started. Yeah. Once you've started typing in the box. But that's not my fault. That's forever. But yeah.

SPEAKER_6

[01:17:18] Things. Things. That's very good. Things. Things. Things do. Things do happen. And it.

KIRAN

[01:17:24] And it actually runs faster than you can press the buttons. But also it does work on a phone. It's reduced. I don't really care about the fact that it works on a phone.

PERSON 1

[01:17:35] Have you tested it? I guess probably not. Because like. Actually. You can kind of do this with React. With. If you think React forgets pages. What? Turn it down. I'm putting. Encoding the. The text. Inside the URL. So you can send people links. Text. But it might as well just send them the program. I don't know. What's that? Or, yeah. You can just send them a single link.

PERSON 3

[01:17:56] Yeah.

PERSON 1

[01:17:57] I guess. Yeah. I guess that could be an option. I'm just worried that feature too much.

PERSON 4

[01:18:00] So if I click. If I. No. But like. It's kind of easy to. No. That wouldn't be hard. This is Monaco, or what?

KIRAN

[01:18:07] Uh, no. It's. I couldn't get Monaco to work.

KIRAN

[01:18:20] I've forgotten what it's called.

PERSON 3

[01:18:23] If you like tab, instead of tabbing in the middle of the page. Let's call it a day now.

KIRAN

[01:18:30] Thank you very much guys, that was really helpful.

PERSON 3

[01:18:33] It's a sticky cursor, it just sort of grabs onto it.

KIRAN

[01:18:36] Yes, thank you. I'm going to end the recording.