Q&A Session for Programming Languages Lecture 21

Session Number: 1206682591
Date: 2020-11-20
Starting time: 14:25

ANON - 14:34
Q: do all statically typed languages have type systems?
Priority: N/A
	Ana L. Milanova - 14:35
	A: Yes, they do.

ANON - 14:36
Q: do dynamically typed langauges have type systems? or only static?
Priority: N/A
	Ana L. Milanova - 14:38
	A: We discuss this during lecture. In short: they both do have some set of rules that we typically refer to as "type systems".
	Ana L. Milanova - 14:38
	A: The "type systems" in statically typed languages are typically a lot more complex.
	Steven Haussmann - 14:39
	A: Whilst, by comparison, dynamic typechecking basically boils down to checking if something is the right type before it's used -- you don't get the same kind of inference and consistency checking
---

ANON - 14:41
Q: why does documentation apply to static types?
Priority: N/A
	Steven Haussmann - 14:41
	A: Dynamic type systems don't include any type information in their source code.

ANON - 14:41
Q: Wouldn't Python be an exception since you can document using type hints?
Priority: N/A
	Ana L. Milanova - 14:42
	A: Type hints are optional. And most programmers don't use them actually. (We just pulled 70K repos from github and only 2K had some annotations.)
	Steven Haussmann - 14:42
	A: They're also not used for compilation or execution -- they're just a standard way to document types that tools can use
---

ANON - 14:42
Q: how are the 3 views of types used altogether?
Priority: N/A
Ana L. Milanova - 14:44
A: These are 3 ways of thinking about types. Usually, we have some combination of these views in mind when we think about types.

ANON - 14:48
Q: is a safe way to tell if a langauge is static or dynamically checked to see if you need to declare a type when having a new variable?
Priority: N/A
Ana L. Milanova - 14:50
A: There is a correlation, but it is not always the case that when there are no type annotations the language is dynamic. Usually this is the case, but not always.
Steven Haussmann - 14:50
A: This would just be part of learning the language. Of course, the compiler will let you know if you need annotations (:

ANON - 14:48
Q: So in Haskell Type Annotations are only optional? Why is that the case?
Priority: N/A
Steven Haussmann - 14:50
A: They can be inferred by the compiler.
Steven Haussmann - 14:51
A: f x = (+ x 1) must take a number and return a number, since its single argument, x, is given to +, and the result of + is returned. So, you don't have to tell it that this function is Num a => a -> a
Ana L. Milanova - 14:52
A: Yes, Haskell does type inference and is able to infer types for many variables and constructs. But this is more complex... There are constructs that it can't infer types for, and for those cases, annotations can help the inference algorithm.

ANON - 14:54
Q: are forbidden errors similar to compiler or runtime errors?
Priority: N/A
Ana L. Milanova - 14:55
A: Forbidden errors are semantic errors. The language designates certain errors as forbidden errors, and then designs a system to prevent those errors.
Ana L. Milanova - 14:56
A: They can be prevented statically, i.e., they will turn into

compiler errors. Or they can be prevented dynamically, i.e., they will turn into runtime exceptions.

Steven Haussmann - 15:02

A: Generally, though, the "forbidden error" itself is something that happens at runtime -- the program itself has to be well-formed enough to run in the first place.

ANON - 15:02
Q: Will Pascal catch all array index out-of-bounds errors at compile time?
Priority: N/A

Ana L. Milanova - 15:04

A: Yes, that is the intention. That means that the type system has to prove that the array access is within bounds. And in order to be able to prove, it will end up restrictive programs significantly. E.g., we have to write if 0 <= i < Bound then a[i] = ... Or there will be some compiler generated code that adds those checks.

Ana L. Milanova - 15:05

A: "will end up restricting programs"

Steven Haussmann - 15:05

A: No, invalid indexing is still a runtime error -- the compiler can just generate code that checks that you're in-bounds.

Steven Haussmann - 15:06

A: (well, at least, some invalid indexing is!) I see that you can also make the compiler demand you do the range-checking yourself (:

ANON - 15:03
Q: Sometimes it is possible to access an index that is "out of bounds" in C / C++, but usually the read is "garbage memory". Is this possible because the bound is not a forbidden error in C/C++?
Priority: N/A

Steven Haussmann - 15:03

A: Yes. In C/C++, an out-of-bounds read causes undefined behavior, which means that no guarantees are made about what happens. The language does nothing to prevent this.

ANON - 15:05
Q: what's an example of how C++ can call quack on a non-Duck object?
Priority: N/A

Ana L. Milanova - 15:06

A: This slide, slide 16 gives an example.

Ana L. Milanova - 15:07

A: It is not a quack, but it is an analogous example. We are calling method foo(int) on an object that does not implement this method.

ANON - 15:09
Q: So the Downcasts do not suceed and so B* q points to an A Object?
Priority: N/A
Steven Haussmann - 15:17
A: The downcast from A to B occurs even if the object isn't actually a B, resulting in incorrect behavior
Ana L. Milanova - 15:17
A: I think you are referring to what happens in Java? Yes, in Java, the downcast will result in a ClassCastException.

ANON - 15:18
Q: If the Downcast happens, then why does q end up invoking A's methods?
Priority: N/A
Steven Haussmann - 15:19
A: Virtual calls always call a function on the actual object; the static type doesn't matter. The problem here is that we have a superclass that lacks a method that the subclass has; the compiler thinks it's safe to call the function, since B has it.

ANON - 15:24
Q: can the program tell structural equivalence if the two types have different names but same constructors?
Priority: N/A
Ana L. Milanova - 15:26
A: In most languages (we get to this a bit later in lecture) the field names are part of the struct type. So if they have different names, even if the same type, they won't be structurally equivalent.

ANON - 15:25
Q: The struct cell on Slide 25 is not a new type or alias for struct but rather the name for the instance of the struct type?
Priority: N/A
Steven Haussmann - 15:29
A: It's defining a new type called "struct type", which is made up of whatever's in the braces. I wouldn't really say it's an instance of a "struct type", since struct, itself, is just a keyword
Steven Haussmann - 15:29
A: err, called "struct cell"

ANON - 15:28
Q: So I'm confused. Are the Field Names checked for Structural Equivalence? If yes, then same type with different names not equivalent, but if no then they would be equivalent right?

Priority: N/A

Ana L. Milanova - 15:31

A: Typically fields names _are_ part of the struct constructed type. Sorry I mistyped in the answer to the previous question. They are checked for structural equivalence.

ANON - 15:30

Q: So what is the difference b/w typedef struct {} cell and struct cell {}? They both create aliases?

Priority: N/A

Steven Haussmann - 15:31

A: The former produces a type named "cell"; the latter produces a type named "struct cell". You generally see the former, since it lets you omit the struct keyword

---

ANON - 15:31

Q: So in the Last Example, the Answer is that they are not Structurally Equivalent b/c different names?

Priority: N/A

Ana L. Milanova - 15:32

A: Yes, that is correct.

ANON - 15:32

Q: what are the two arrays not name equivalent since they are both called array?

Priority: N/A

Ana L. Milanova - 15:33

A: I am assuming you are on slides 28-29. They are not Name equivalent, because each one is a different array type object. They are Structurally equivalent.

ANON - 15:34

Q: i don't really understand how programs check for structural equivlance? is it based off of a tree of attributes?

Priority: N/A

Ana L. Milanova - 15:36

A: Yes, essentially. We will apply the recursive definition. Two types are structurally equivalent if they are made up by the same type constructor applied on structurally equivalent arguments.

ANON - 15:34

Q: Follow up: So the "blue" and "red" are object definitions?

Priority: N/A

Ana L. Milanova - 15:37

A: These are "type objects", yes, you can think of it this

way, we are creating these two different "type objects". And they stand for different types under name equivalence.

ANON - 15:37
Q: for slide 29, how come the arrays are not name equivlent?
Priority: N/A
Ana L. Milanova - 15:39
A: Because the red and blue arrays are two different types under Name equivalence.

ANON - 15:45
Q: how does the examples on slide 37 illustrate type equivalence?
Priority: N/A

ANON - 15:48
Q: So if we used Structural Equivalence, then Q would be included with P, R in the same EQ Class?
Priority: N/A
Ana L. Milanova - 15:51
A: I want to double check the example on Slide 37 to make sure I have everything stated correctly. I'll answer your question later before I post the Q&A, this is a longer answer in any case!
Ana Milanova - ???
A: The equivalence classes on Slide 37 are correct. Regarding the question about the pointer types. P and Q are not equivalent, i.e., they are in different equiv. classes. The equivalence check is as follows: two pointer types P1 and P2 are equivalent if they are made of _equivalent_ components. The types P1 and P2 point to, respectively, must be equivalent according to the rules of equivalence for those types. Since in our case P and Q point to struct types, we are looking for the loose name equivalence of structs. But P points to a struct A and Q points to a struct B, and they are not equivalent. Therefore, P and Q will not be equivalent.

ANON - 15:50
Q: Oh so here, A is an alias for B, so a = b is valid.
Priority: N/A
Ana L. Milanova - 15:51
A: Yes, that is correct.

ANON - 15:50
Q: How are A and B equivalent? I thought typedef struct B A would make a different A and so none of the statments should succeed
Priority: N/A
Ana L. Milanova - 15:53

A: typedef just introduces an alias, a different name for a type. The struct B type is constructed via the struct type constructor. And here we introduce A, a new name, or an alias to that same struct type object.

ANON - 15:54
Q: how does coercion work in C just to change the type?
Priority: N/A

Ana L. Milanova - 16:05

A: This is implicit conversion. The compiler implicitly converts when there is no loss of precision, e.g., we can implicitly convert an int to a float.

ANON - 16:00
Q: Does typedef inroduce an Alias even with Function Pointers? The syntax is slightly different
Priority: N/A

Ana L. Milanova - 16:09

A: Yes, you are right, the syntax is slightly different. It does introduce an alias again, to that function type. The way I think about this is: we create the function type as part of the typedef and we name it accordingly.

ANON - 16:00
Q: I'm very confused about slides 33 & 34; why aren't p,q,t equivalent? why are the "pointer" colors different?
Priority: N/A

Ana L. Milanova - 16:26

A: Each "pointer to cell" type is a distinct "pointer to cell" type under Name equivalence.

ANON - 16:02
Q: so p,q,t are types? not instantiations of types?
Priority: N/A

Ana L. Milanova - 16:27

A: Sorry this wasn't explained clearly. Here p,q,t,r,s, etc. are the program variables, which we declare to be of those types.

ANON - 16:02
Q: what language syntax is this?
Priority: N/A

Ana L. Milanova - 16:31

A: I don't want to claim a particular language, I might make a mistake. It follows that pseudocode syntax we have been using, type n = T is a type declaration and x : ... is a variable declaration.

ANON - 16:03
Q: thank you! i get it
Priority: N/A

ANON - 16:05
Q: Under the hood, aren't pointers and arrays the same thing? (in C, at least)
Priority: N/A
Steven Haussmann - 16:11
A: They're the same general concept, yes -- an array is a blob of memory with many instances of the array's type stuck together.

ANON - 16:05
Q: Does Haskell use Implicit Type Conversion b/w String + Name? I think I did an Operation b/w these in Homework 6 and was wondering if this was how it worked. Also how does Explicit Type Conversion work in Haskell? Is it just :: (New Type)?
Priority: N/A
Ana Milanova - ???
A: Name was an alias to the String type in our homework, so there should be no type conversion. I don't think Haskell has a way of "casting" from one type to another as there is a cast in C. Even if there is a way to do that, this is not usually the right way to go. Haskell can convert from one type to another, e.g., Int to String using show, or from Float to Int using round, or trunc. The latter does lose precision.

ANON - 16:08
Q: Structs trees/graphs can have one child or multiple children?
Priority: N/A
Ana L. Milanova - 16:12
A: Typically we have more than one child. It depends on the constructor and its arguments. For example, struct can have many fields, and they will show up as children in the tree.

ANON - 16:11
Q: how do we know how to draw the type tree?
Priority: N/A
Steven Haussmann - 16:15
A: The tree tells you what a type is made up of. You go through each element of the struct and draw their own type trees. Scalar types (i.e. types that aren't made up of other stuff) are leaves.

ANON - 16:11
Q: When you say function pointers introduce an alias to that function type? What does that mean? It feels like it's not clear what it is aliasing to me.
Priority: N/A
Ana L. Milanova - 16:35
A: The typedef, e.g., typedef int (*F) (int); does 2 things. It creates a new type, the pointer-to-an-int-to-int-function, then names that type F. We can think of that new type as having some compiler name, Anon, and typedef making F and Anon aliases.

ANON - 16:12
Q: Oh I see. The Function Type can refer to the Input + Output Parameters.
Priority: N/A
Ana L. Milanova - 16:35
A: Yes, the function type is defined in terms of parameter types and return type.

ANON - 16:13
Q: what is the program on this slide illustrating?
Priority: N/A
Ana L. Milanova - 16:36
A: Advanced C programming. What I wanted to illustrate is how C programmers use function pointers to emulate OO functionality (like dynamic dispatch). But we didn't have time for that discussion.

ANON - 16:16
Q: Why did the Last Example need to Type Cast f By (struct_chunk* (*) ()) f as opposed to just (struct_chunk*) f
Priority: N/A
Steven Haussmann - 16:17
A: f is a function pointer, not just a pointer to a struck_chunk

---

ANON - 16:19
Q: will we be required to write C in Homework 7?
Priority: N/A
Ana L. Milanova - 16:22
A: There will be no programming, but there will be questions on C types --- making sense of C types and drawing the type trees.

ANON - 16:19
Q: So it casts it to the Function Pointer (*) that takes no Arguments

and Returns struct_chunk∗, But what does (∗) mean here, it is a Function Pointer without any Name to A Function?
Priority: N/A

Ana L. Milanova - 16:25

A: Yes, (∗) is this anonymous function pointer type. The expression casts f, which was a void ∗, to that function pointer type. (And function pointer type equivalence is structural equivalence.)