Programming Languages

MUPL

WashU CSE 425s
Prof. Dennis Cosgrove
#17: Wed, Nov 02, 2022

(& S Q) When we implement a programming language, do we have to determine if it's functional or oop at the first place?

- you have to decide what your language will support?
- will it have classes? subclassing? private instance variables? immutable values? first class functions? arrays? for loops? operator overloading?
- how will the programmer express these things in your language?

alternatively, you could decide do I care more about speed than anything else? do I care that it runs on the web over everything else.

(& S Q) How would an interpreter handle structs? Treating an environment as a variable?

For a language like C, there would be a FieldAccessExpression in the AST

color.red

FieldAccessExpresion(Binding(“color”), “red”)

S&Q: Can you explain the difference between compilers and interpreters again? I didn't understand his analogies.

S&Q: I was a little confused by compiler and interpreter distinction? Could you explain this in a different way than he does?

Interpreters evaluate a program.

Compilers translate a program to another language.

- notes-and-tips

S&Q: compiler and interpreter

```
function line(m, x, b) {
    return m*x+b;
}

slope = 2
offset = 4
y = line(slope,3,offset)
```

Interpreters evaluate a program.

Compilers translate a program to another language.

S&Q: I was a little confused by compiler and interpreter distinction? Could you explain this in a different way than he does?

Interpreters evaluate a program.
Compilers translate a program to another language.

- notes-and-tips

*** C?
*** Java?
*** Python?

(& S Q): What's the difference between a compiler language and a language that use intepreter?

in theory, nothing. it is an implementation issue.

in practice, ~~interpreted languages~~ languages whose default runtime implementation is an interpreter tend to be more flexible.

(& S Q): Is Assembly the "metalanguage" for C?

C is portable assembly language by design

C is (what do you know?) well translated to assembly

(& S Q): Evidently, there is a trend where compiled languages are faster than interpreted languages (or so I've been told). If the fact that they are compiled or not is not the reason for this trend, then what is?

- compilers tend to compile into ASSEMBLY language
- compiled languages tend to be more constrained
  - int + int is a single instruction in C
  - one can add methods to 4, replace +, do whatever you want in Ruby

`final class Integer` storytime

S&Q: Interpreted C?!?

I still don't really the comparison and explanations about complier and interpreter? What does that mean there is no a "complied language"? Does that mean in theory we can have an interpreter for C?

- not just in theory: https://www.softintegration.com/

S&Q: When would you want an interpreter vs compiler?

(& S Q): Can you explain a bit more about Java compilers?

(& S Q): How do languages use both a compiler and interpreter? I've only heard of a compiler so I'm unsure what the difference is, especially if both are used like in Java.

Java

Java compiles to machine independent Java bytecode (intermediate language for the Java Virtual Machine)

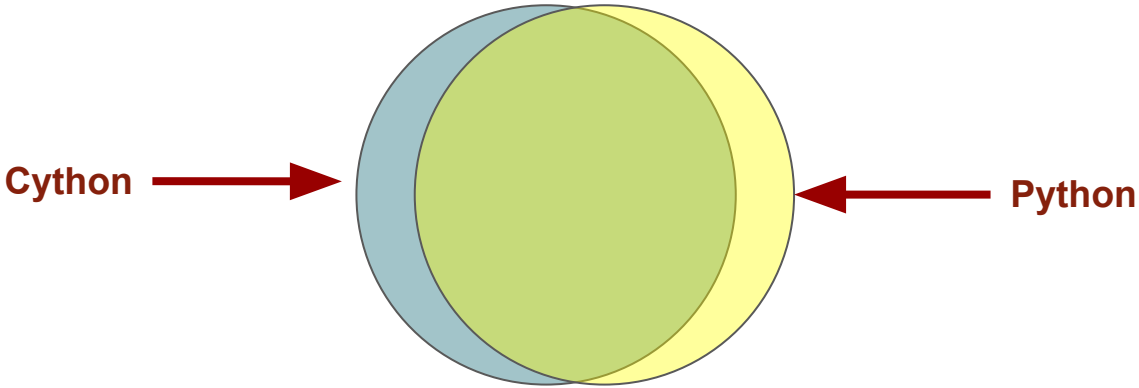
The Java Runtime Environment runs that program (a lot of leeway)

Just-In-Time compiler does runtime compilation of into machine language
“identifies parts of the code where the speedup gained from compilation or recompilation would outweigh the overhead of compiling that code.”

Is it possible to compile a traditionally interpreted language (like compiling Python to machine code)? Is this what Cython does?

https://cython.org/

- compiles most of Python to C code (which in turn is compiled)
- while we're here… adds capability to call C directly



(& S Q) What the difference is between what the interpreter should handle gracefully and what is allowed to cause it to crash?

I'm a bit confused on how closures will be treated in MUPL. Are they also going to be built as structs?

Could you explain how closures are evaluated in MUPL?

Which expressions are evaluated recursively, and in what environments?

(& S Q) When passing environment, are we passing the value for that expression or what? Why can't we just call with eval-exp with expression being set to that value?

S&Q: I've always wondered - how were the first programming languages interpreted/compiled? What is there to compile to if you're first? How do you build an interpreter in another language A if there is no other language?

(& S Q) I know the basis of most programming languages is C but what language was used to first create C. What was the original creator of programming languages? How could you make a programming language without an original programming language?

S&Q: Can you interpret a language A using language A itself?

Lisp did in 1962 https://en.wikipedia.org/wiki/Bootstrapping_(compilers)#History

https://en.wikipedia.org/wiki/Bootstrapping_(compilers)#List_of_languages_having_self-hosting_compilers

Let's be real, how hard is building our own Interpreter?

Not that hard (in my opinion)

I believe IDE support is a larger effort than the core language

How much effort do I need to put in before the community starts building all of the libraries?

Kotlin

https://kotlinlang.org/

generates to Java bytecode

start off your language with all of Java’s libraries

Jython Storytime

generates Java byte code directly

access (even extend Java classes)

https://en.wikipedia.org/wiki/Jim_Hugunin

(& S Q): Can you review closures and environments a little more?

The evaluation of a function is a value with two parts.

1) The code itself

2) The lexical scope environment (can be the subset which is used by the code)

this will become more clear when you complete MUPL

Note: the language will have rules for how to handle lookup order of the many environments

(& S Q): could you give an example of a closure? or at least explain it further

Closures are values: (the function * the environment)

```
Future<Void>[] futures = new Future[words.length];
String[] translatedWords = new String[words.length];
for (int i = 0; i < words.length; ++i) {
    int _i = i;
    futures[i] = void_fork(() -> {
        translatedWords[_i] = translator.apply(words[_i]);
    });
}
join(futures);
return translatedWords;
```

Objects in Java are references

```
Future<Void>[] futures = new Future[words.length];
String[] translatedWords = new String[words.length];
for (int i = 0; i < words.length; ++i) {
    int _i = i;
    futures[i] = void_fork(() -> {
        translatedWords[_i] = translator.apply(words[_i]);
    });
}
join(futures);
return translatedWords;
```

```
Future<Void>[] futures = new Future[words.length];
String[] translatedWords = new String[words.length];
int[] array = { 0 };
for (array[0] = 0; array[0] < words.length; ++array[0]) {
    futures[array[0]] = void_fork(() -> {
        translatedWords[array[0]] = translator.apply(words[array[0]]);
    });
}
join(futures);
return translatedWords;
```

(& S Q) How far off is the way we're supposed to implement closures from how real programming languages implement closures?

- the required subset of the environment would be stored
  challenge problem

Do you have any hints or suggestions for the MUPL studio and assignment?

```
(struct var  (string) #:transparent)  ;; a variable, e.g., (var "foo")
(struct int  (num)    #:transparent)  ;; a constant number, e.g., (int 17)
(struct add  (e1 e2)  #:transparent)  ;; add two expressions
(struct ifgreater (e1 e2 e3 e4)    #:transparent) ;; if e1 > e2 then e3 else e4
(struct fun  (nameopt formal body) #:transparent) ;; a recursive(?) 1-argument function
(struct call (funexp actual)       #:transparent) ;; function call
(struct mlet (var e body) #:transparent) ;; a local binding (let var = e in body)
(struct apair (e1 e2)     #:transparent) ;; make a new pair
(struct fst  (e)    #:transparent) ;; get first part of a pair
(struct snd  (e)    #:transparent) ;; get second part of a pair
(struct aunit ()    #:transparent) ;; unit value -- good for ending a list
(struct isaunit (e) #:transparent) ;; evaluate to 1 if e is unit else 0
```

MUPL Rosetta Stone

```
val mlet_add_example =
   let
       val a = 425
   in
       a + 231
   end

val apair_example = (425, 231)
val fst_example = #1 (425, 231)
val snd_example = #2 (425, 231)

val ifgreater_example = if 425 > 231 then 42 else 7
```

```
(define mlet_add_example
       (mlet "a" (int 425) (add (var "a") (int 231))))

(define apair_example
       (apair (int 425) (int 231)))
(define fst_example
       (fst (apair (int 425) (int 231))))
(define snd_example
       (snd (apair (int 425) (int 231))))

(define ifgreater_example
       (ifgreater (int 425) (int 231) (int 42) (int 7)))
```

(& S Q): Can you go over ASTs a little more?

Eclipse's Java AST:
https://www.ibm.com/docs/api/v1/content/SS5JSH_9.5.0/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-tree.html

(& S Q): Do all interpreters use ASTs? Is using a list of pairs for vars actually efficient?

Lab 5: MUPL

- added clarification tests
- (optional) mupl-value? and expand-environment
- required MUPL Programs mupl-map too big of a first step?

Road To Victory

1. UW MUPL Assignment Part 1 Warm Up
2. WashU Optional mupl-value?
3. WashU Optional expand-environment
4. UW MUPL Assignment Part 2 Implementing the MUPL Language
5. UW MUPL Assignment Part 3 Expanding the Language
6. WashU Required MUPL Programs Exercise
7. UW MUPL Assignment Part 4 Using The Language mupl-map, mupl-mapAddN
8. UW MUPL Assignment Part 5 Challenge Problem

MUPL Studio Storytime From Piazza From The Past

student:
Is MUPL Studio Required?
I noticed that the MUPL Studio is located in the hw5 folder. Does this mean this studio is similar to the Thunks and Streams studio in that we only need to do it to help out with the lab?

prof:
for thunk and streams, i could foresee someone struggling and still be able to build hw4 so I downgraded it to a warm up.

it is hard for me to imagine someone being able to build mupl-map and struggling with mupl-double and friends, so I believe it is in everyone's best interests to keep it a required studio.

sound fair?

when you have built the studio and mupl-map, let me know if you agree, please.

student:
I just finished the studio and lab and I definitely agree.

(& S Q): Confused on the "skipping parsing" part

MUPL: Why Check For Some Errors But Not All?

I don't really understand the reasoning behind the statement/distinction that "it's OK" to allow Racket to throw weird errors when the data types don't match the things our struct expects, but it's not OK for more complex expressions, where only by evaluating part of it will we know that it's wrong.
***What** is so different, conceptually, between const e and negate e that we have to typecheck one at the interpreter level but not the other?*
*And/or perhaps I don't understand what makes for a "valid" vs. an "invalid" AST in our created language. **why** is one case valid and the other not?*

- it is not really **`const e`** it is more like **`const integer-literal`**
- some other part of the system (parser?) would have already determined that const(“fred”) was an invalid MUPL program before handing it off to the interpreter

(& S Q): We can assume that the initial input tree is valid, but every recursive step above that needs to be checked? Is that correct?

MUPL: Why Check For Some Errors But Not All?

imagine:

```
(define ast (parse source))
(interpret ast)
```

the parser would reject

```
(const #t)
(add (const 3) “uh-oh”)
(negate -1)
```

but would not reject a valid ast which an expression evaluated to the wrong type

(& S Q): Why not go the extra step of handling illegal ASTs? Wouldn't this make debugging better for the programmer?

this could/should be done at AST construction time.

(& S Q) I admit a lot of this went over my head and I need to re-visit the material.

dig into the assignment and then (if you get stuck) revisit the videos???

Is this or context an example of the environment? Is this a workaround in java?

When compiling into a language without closures, we need each function to take an extra parameter for the environment. Is this the reason for "public static void main(String args[])" in Java or "main(argc, char* argv[])" in C/C++, or is this something different?

S&Q: OOP Racket?

Racket structs seem a little closer to object-oriented programming than SML's algebraic datatypes. How much further in this direction can we go with Racket? For instance, can we extend one Racket struct to make another struct with more attributes? Given the dynamic typing, can we make variables of the "sub-struct" type work with functions designed for elements of the previous (non-extended) struct? I see why things like this wouldn't fit in well with SML's paradigm of everything having one well-defined type, but Racket seems like it could possibly give a nice way to combine functional and object-oriented style. Do Racket programmers do things like this much in practice?

1) you can always build everything that can be built in anything
2) https://docs.racket-lang.org/reference/mzlib_class.html
3) Lisp vs. SML, which is easier to to build dynamic dispatch and overriding? I would agree it would feel as though you would be fighting Lisp less than SML… especially given how strict SML is

Will there be warm-up or in-class exercises using structs? I think some practice would be good before we dive into the MUPL assignment.

Why SML->Racket instead of Racket->SML?

Why did this course start with SML and not Racket? it seems like the learning curve would have been much more manageable starting with Racket:

- Dr. Racket is much more user-friendly, so we can learn functional-programming concepts in a more friendly setting before diving into SML
- We can also get practice on functional concepts (closures, no or little mutation, etc.) before having to deal with SML-specific but still very new and weird concepts like type inference and pattern matching, instead of lumping it all together
- Certain elements of Racket, like having actual multi-parameter functions separate from the concept of currying, would let us learn currying in a setting where it's a new "additional" thing, and then when we *start* SML, **from the beginning**, we could just go "actually, in SML, there are only single-parameter functions and currying" without having to start with a lie and then backtrack from it.

Also heard in office hours: *“I have found Racket to be much easier than SML”*

S&Q: How Do Hybrids Like IronPython Work

So this is only sort of a question, but I think it would be really interesting to see how modern languages use hybrid interpret/compile strategies to be more efficient. Like I know that javascript is slow, but because it uses V8 within the browser it is fast- but what is V8, and what does it do? What is IronPython vs Cython vs Python? Etc. etc. etc. Grossman hinted at what java is doing, but getting an in depth study of how these optimizations work in a real-world scenario would be super cool I think.

https://en.wikipedia.org/wiki/Just-in-time_compilation

https://en.wikipedia.org/wiki/Jim_Hugunin