Comparative Analysis of Multi Paradigms Languages

Faisal Mumtaz
i16-1024
MS(CS)

Faisal Mumtaz
i16-1024
MS(CS)

Umar Munir
i16-1011
MS(CS)

Shahran Gohar
i16-1024
MS(CS)

Mehreen Alam
i16-1024
MS(CS)

Shahid Hussain
i16-1024
MS(CS)

ABSTRACT

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

Compiled/interpreted: The difference lies not in the language but how the language has been implemented. A compiler translates the source code of the program into another language format that can be directly executed by a lower-level machine. This can be an abstract machine (such as .NET or the Java Virtual Machine) or the actual machine. In the latter case, the language format that is the target of the compiler is machine code. The translation from source code into lower-level code depends on the abstract syntax and on the operational semantics of the programming language. An interpreter executes the source code directly; informally, it may help to think of the interpreter as executing the program line by line. A more correct understanding is that the interpreter walks through the abstract syntax tree generated by the parser and executes each node in this tree. If a node is a leaf, the leaf is executed. If a node is an internal node, each sub-tree is visited and executed. Exactly how this is to be done depends on the abstract syntax and on the underlying semantics of the programming language. Out of the many programming languages in this world, some of them are called compiled languages while some are interpreted. For compilation, the software uses is called compiler while for interpreter is used for interpreted language. For a compiled language, an interpreter can be built but the reverse is impossible. That is, all the interpreted languages cannot be a compiled language. Additionally, being interpreted or compiled is not the property of the programming languages, but the design of some languages make them unsuitable for native code generation. Assertion: specifies that a program satisfies certain conditions at particular points in its execution. An assertion violation indicates a bug in the program. Thus, assertions are an effective means of improving the reliability of programs and function as a systematic debugging tool. There are three types of assertion: pre-conditions, post-conditions and invariants. Preconditions specify conditions at the start of a function; post-conditions specify conditions at the end of a function while invariants specify conditions over a defined region of a program. Asserts are to be used primarily for checking parameter types, classes, or values, checking data structure invariants, checking "can't happen" situations (duplicates in a list, contradictory state variables) or after calling a function, to make sure that its return is reasonable. However, asserts are not to be used for handling run-time errors, like entering a negative number when positive is needed. But used to catching the program errors.

2. RELATED WORK

3. FEATURES

3.1 Bound Checking

In computer programming, bound checking is any method of whether variable detecting variable is within bound before it is used. A failed bounds check usually results in the generation of some sort of exception signal.

3.1.1 Range checking

It is usually used to check that whether a number fits into a given type. A range check is a check to make sure a number is within a certain range; for example, range check will ensure that a value that will assign to a 16-bit integer is within the capacity of a 16-bit integer. Some range checks may be more restrictive; for example, a variable to hold the number of a calendar month may be declared to accept only the range 1 to 12

3.1.2 Index checking

In index checking a variable being used as an array index is within the bounds of the array. Index checking means all expressions indexing an array, the index value is checked

Table 1: Index Checking

	Index checking	Range checking
Scala	✔	✔ (statically check)
Swift	✔	✔
F#	✔	✔
Rust	✔ (at run time)	-
Vb.net	✔	✔
C#	✔	✔
D	✖	✔
Oz	-	-
Matlab	✔	✔(statically check)
Python	✔	✔

against the bounds of the array, which were created when the array was defined, and if the index is out-of-bounds, an error occur and further execution is suspended. If a number outside of the upper range is used in an array, it may cause the program to crash, or may introduce security vulnerabilities, index checking is a part of many high-level languages.

3.1.3 Examples

3.1.4 Scala

Array representation in scala
scala> val a1 = Array(1, 2, 3)
a1: Array[Int] = Array(1, 2, 3)

3.1.5 Swift-range checking

func contains(Bound) Returns a Boolean value indicating whether the given element is contained within the range.

3.2 Type Safety

The compiler will validate types and through an error if you assign a wrong type to a variable. Type safety is checking for matched data types during compile time. For example, int a ="John" returns error as variable 'a' is an integer and we are assigning a string value. These data type mismatches are checked during compile time. Type safe code can access only the memory locations that it has permission to execute. Type safe code can never access any private members of an object. Type safe code ensures that objects are isolated from each other and are therefore safe for inadvertent or malicious corruption

3.2.1 The advantages type safety

At compile time, we get an error when a type instance is being assigned to an incompatible type; hence preventing an error at runtime. So at compilation time itself, developers come to know such errors and code will be modified to correct the mistake. So developers get more confidence in their code. Run time type safety ensures, we don't get strange memory exceptions and inconsistent behavior in the application.

3.2.2 scala

Scala is strongly type and smart about static type. Scala has powerful type inference. It will figure out itself mostly no need to tell it the types of your variables.

3.2.3 Swift

Swift is type safe, it performs type checks when compiling code and flags any mismatched types as errors. This help in early catch and fix error in the development process. It provides type inference which basically means that coders donâĂŹt require to spend more time in defining what types of variables they are using.

3.2.4 F#

In f#, static type checking can use almost as an instant unit test âĂŞ making sure that your code is correct at compile time. F# is more type-safe than C#, and how the F# compiler can catch errors that would only be detected at runtime in C#.

3.2.5 Rust

Rust is a type-safe language. Rust has an escape valve from the safety rules. When you absolutely have to use a raw pointer. This is called unsafe code, and while most Rust programs dont need it, how to use it and how it fits into Rustś overall safety scheme in
https://www.safaribooksonline.com/library/view/programming-rust/9781491927274/ch21.html#unsafe-code

3.2.6 VB.net

Type safety in .NET has been introduced to prevent the objects of one type from peeking into the memory assigned for the other object.

3.2.7 C#

Type safety prevents assigning a type to another type when are not compatible. public class Employee

public class Student In the above example, Employee and Student are two incompatible types. We cannot assign an object of employee class to Student class variable. If you try doing so, you will get an error during the compilation process. Type safety check happens at compile time it's called static type checking Example Cannot implicitly convert type 'Program.Employee' to 'Program.Student'. . When tried to type cast object of wrong type. We get Unable to cast object of type âĂŸfirst objectâĂŹ to âĂŸsecond objectâĂŹ type checking happens at runtime, hence it is called runtime type checking

3.2.8 D

D has compile-time type safety.

3.2.9 OZ

OZ also known as MOZART. Oz variables are single-assignment variables or more appropriately logic variables. A single assignment variable has a number of phases in its lifetime. Initially it is introduced with unknown value, and later it might be assigned a value, in which case the variable becomes bound. Once a variable is bound, it cannot itself be changed.

3.2.10 Matlab

MATLAB is a loosely or weakly-typed language. Difference between MATLAB and a strongly-typed language is that you don't have to explicitly declare the types of the variables you use. For example, the declarations x=5; x='foo'

Table 2: Type Safety

Languages	Type Safety
Scala	Strongly type, Static type, powerful type Inference
Swift	Type check at compile time, Support type inference
F#	Static Type Checking, Compile Time
Rust	Type Safe, escape valve, unsafe to use raw pointers
VB.net	Type safety use for memory security
C#	Static type checking, type checking compile time
D	Type safe, compile time
Oz	Single Assignment variables, Once value is assigned to variable it can never be change
Matlab	Weakly type language, no need to assign type explicitly,
Python	Dynamically type language,

immediately following one another are perfectly acceptable; the first declaration causes x to be treated as a number, the second changes its treatment to a string

3.2.11 Python

Python or Ruby are often referred to as dynamically typed languages, which throw exceptions to signal type errors occurring during execution

3.3 Exception Handling

An exception handler is a block of code that is executed if an exception occurs during the execution of some other block of code. In this sense, exceptions are a kind of control statement. Raising an exception transfers the flow-of-control to exception handling code. User can also throw own created exception.

3.4 Meta-programming

Meta-programming is the capability to adapt itself (meta stack overflow which is the place to ask question about stack overflow itself). We can also say the ability to treat programs as their data. It means that a program can be designed to read, generate, analyze or transform other programs, and even modify itself while running. Meta-programming is not one specific technique, but rather an ensemble of concepts and techniques. There are two different ways of doing metaprogramming: on the Syntax level and at Runtime. ection explain these components.

- **Features for Syntax:** These are feature of languages through which Syntax meta-programming apply.
- **Features for Runtime:** These are feature of languages through which Runtime meta-programming apply.

Reflection: is the ability of a computer program to examine, introspect, and modify its own structure and behavior

Lang.	Throw	Handler	Assertion
Sca la	throw	try { instructions } catch (exception) { instructions} finally{instructions}	Assert(stat ement)
Swi ft	throw exce ption ()	do { try expression ... instructions } catch exception { instructions }	assert(condi tion,descrip tion)
Falc on	raise excep tion	falcon.HTTPError (status,title=None, description= None)	
F#	raise exce ption	try expression with pattern or try expression finally expression	assert cond ition

Rust	Err(excep tion)	match fun_nam(x,y)
		{
		Ok(v) =>{
		println!("{}", v); },
		Err(err) =>{
		println!("{}", err);
		}}	

Vb .Net	throw excep tion	Try instructions Catch exception When condition instructions ... Finally instructions End Try	Debug.Ass ert(condi tion)
C#	throw excep tion;	try { instructions } catch (exception) { instructions} finally {instructions}	Debug.Ass ert(condi tion);
D	throw	try { instructions } catch (exception) { instructions} finally {instructions}	
Oz	{exception. 'raise'X}	try S catch Pattern_1 then S1 Pattern_2 then S2 finally S_final end	
Mat lab	throw(excep tion)	try tab statements catch exception tab statements end	assertError (assert able,actual, identifier)

R	throw()	tryCatch({
		expr},
		war=function(w){
		warning-handler-
		code},
		error = function(e)
		{error-handler-
		code}, finally = {
		cleanup-code
		}
			assertError (expr,verbo se=FALSE)

Pyth on	raise excep tion	try:
		Tab instructions
		except exception:
		Tab instructions
		else:
		Tab instructions
		finally:
		Tab instruction
			assert cond ition

Table 3: Exception Handling syntax in different lan-

Prog. Lang.	Ways of Doing		Features for compile time	Features for Run Time
	Compile Time	Run Time		
R		✓		Objects
Scala	✓	✓	Reflection	Macros
Swift	✓			Templates
Falcon	✓			Macro
F#	✓	✓		Quotation
Rust	✓			Macros
VB.net	✓	✓	Reflection	Reflection
C#		✓		Objects
D	✓			Template
Oz	x	x		
Matlab	x	x		
Python		✓		meta-classes

Table 4: Meta programming features in different languages

at runtime.

3.5 compiled / interpreted

3.6 Assertion

3.7 conditional compilation Shaad

3.8 file handling Shaad

3.9 Imutable

In Muti-paradigm programming languages, an immutable object is an object whose state cannot be modified after it is created. This is in contrast to a mutable object (changeable object), which can be modified after it is created. In some cases, an object is considered immutable even if some internally used attributes change but the object's state appears to be unchanging from an external point of view. For example, an object that uses memoization to cache the results of expensive computations could still be considered an immutable object.

3.10 Mutable

A mutable object, by contrast, has data fields that can be altered. One or more of its methods will change the contents of the object, or it has a Property that, when written into, will change the value of the object. If you have a mutable object- the most similar one to String is StringBuffer inC#- then you have to make a copy of it if you want to be absolutely sure it won't change out from under you. This is why mutable objects are dangerous to use as keys into any form of Dictionary or set- the objects themselves could change, and the data structure would have no way of knowing, leading to corrupt data that would, eventually, crash your program. However, you can change its contents- so it's much, much more memory efficient than making a complete copy because you wanted to change a single character, or something similar. Generally, the right thing to do is use mutable objects while you're creating something, and immutable objects once you're done. This applies to objects that have immutable forms, of course; most of the collections don't. It's often useful to provide read-only forms of collections, though, which is the equivalent of immutable, when sending the internal state of your collection to other contexts- otherwise, something could take that return value, do something to it, and corrupt your data

3.11 imperative control Shahid

3.12 Explicit concurrency Shahid

4. DISCUSSION AND ANALYSIS

Compiled/interpreted: Major advantage of compilation is the fast performance as it directly used the native code of the target machine and hence has the opportunity to apply quite powerful optimizations during the compile stage. Since the translation is done only once during the compilation, program only needs to be loaded and executed. Major advantage of interpreted is that ease of implementing logic especially for dynamic languages. Also there is no need to compile code and the programs can be executed directly. It is also easier to debug since programs can be executed side by side. Keeping this in mind, compiled languages shall be suitable for the intensive parts of an application requiring heavy resource usage whereas less intensive parts could be written in interpreted languages, e.g. interfaces, invoking the application, ad hoc requests or prototyping. Asserts are a useful debugging tool. They help detect errors that might otherwise go undetected, detect errors sooner after they occur and also ensure that the statement about the effects of the code is true. The disadvantage of using asserts is reporting an error where none exists and failing to report a bug that does exist. Asserts are also not side-effect free. They also consume extra time and memory to execute. Assert is different from exception handling as occurrence of the exception may go unnoticed while asserts ensure one gets aware of the bug. Asserts are sometimes referred to as lazy exception handling. Conclusion If the programmer has to choose between speed and ease of programming, then the choice has to be made between languages opting for compiled or interpreted. A language having the facility of asserts provide the programmers with the ease of detecting errors that would have been impossible to catch using regular exception handling.

5. CONCLUSIONS

6. FUTURE WORK

7. ACKNOWLEDGMENTS

7.1 References

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] articleclarke2006historical, title=A historical perspective on runtime assertion checking in software

Table 5: compiled vs interpreted

	compiled	interpreted
Scala	actually a compiled , wherein everything you type gets compiled to the byte code and it runs within the JVM.	illusion of interpreted
Swift	compiled	
Falcon	compiled, The Falcon compiler contains a meta-compiler[23] that supports macro expansions. A Falcon Virtual Machine in the standard compiler drives the meta-compiler. Output generated from the meta-compiler is sent to the language lexer as if part of the original source.	
F#	compiled, open source cross platform compiler from F# Software Foundation	
Rust	compiled. First, the Rust compiler does all the Rust specific stuff like type and borrow checking; in the end, it generates LLVM-IR. IR stands for intermediate representation and it's comparable to assembly, but a tiny bit more high level and most importantly: platform independent.	
Vb .Net	version 6 and above, both compiled and interpreted	interpreted
C#	compiled	
D	compiled	
Oz	yes, Oz code can be compiled into command line executables. The compiled code is not native binary, but a shell script-wrapper with embedded Oz virtual machine bytecode.	yes
Matlab		yes, you can write code and just execute it from the
R	an interface to compiled code, because all key routines are run in compiled code (through .C, .Call., .Internal, .Primitive interfaces, etc.) But does not compile	yes
Python		yes
	Pre-Post conditions	Quantification

Table 6: Assertion

	Pre-Post conditions	Quantification	Pre-State Values	Global Assertions	Language Integration
Scala	no		no	no	import org.scalatest.Assertions._
Swift					
Falcon	yes	yes	no	yes	import falcon
F#	no	no	no	no	open FsUnit [<AbstractClass>][<Sealed>]type Assert =
Rust	yes	yes	yes	yes	
Vb .Net	no	no	no	no	Debug.Assert Method System.Diagnostics Namespace Public NotInheritable Class Assert
C#	no	no	no	no	assert method in class Debug public static class Assert
D	no	no	no	no	no
Oz	yes	yes	yes	yes	export Literals Assert
Matlab	yes	yes	yes	no	yes, python, c,c++, C#, java, fortran
R	yes	yes	yes	yes	assertthat -assert_that() signal an error -see_if() returns a logical value, with the error message as an -validate_that() returns TRUE on success, otherwise returns the error as a str
Python	yes	yes	no	no	assert method
	Pre-Post conditions	Quantification	Pre-State Values	Global Assertions	Language Integration

Table 7: **Imutable programming features in different languages**

Immutable	
Scala	val maxValue = 100
Swift	let (firstNumber, secondNumber) = (10, 42)
Falcon	-
F#	-
Rust	const mut acc = 1
Vb.net	const PI = 1
C#	const PI = 3.14149
D	imutable int len = 1
Oz	-
Matlab	function h = planck % Planck's constant. h = 6.626068e-34; % Units are m^2 kg / s
R	a <- 1 lockBinding("a", globalenv())
Python	class Foo(object): CONST_NAME = "Name"

Table 8: Mutable programming features in different languages

Immutable	
Scala	var maxValue = 100
Swift	var (firstNumber, secondNumber) = (10, 42)
Falcon	array=[1,2,3]
F#	let a = 1
Rust	let a = 1
Vb.net	Dim num1 as Integer = 1
C#	float PI = 3.14149
D	mutable int len = 1
Oz	{Browse {4+2} div 2}
Matlab	h = 6.626068e-34;
R	a<- 1
Python	var a = 1

development, author=Clarke, Lori A and Rosenblum, David S, journal=ACM SIGSOFT Software Engineering Notes, volume=31, number=3, pages=25–37, year=2006, publisher=ACM

[2] articlehoare2003assertions, title=Assertions: A personal perspective, author=Hoare, Charles Antony Richard, journal=IEEE Annals of the History of Computing, volume=25, number=2, pages=14–25, year=2003, publisher=IEEE

[3] articlekamp2012my, title=My compiler does not understand me, author=Kamp, Poul-Henning, journal=Communications of the ACM, volume=55, number=7, pages=51–53, year=2012, publisher=ACM