6
Related Work

In this chapter we review related work, and we split it into two sections: in the first section we review other Lua projects, while in the second section we review other projects that are not related to Lua.

6.1 Other Lua projects

Metalua [Fle07] is a Lua compiler that supports compile-time metaprogramming (CTMP). CTMP is a kind of macro system that allows the programmers to interact with the compiler [FT07]. Metalua extends Lua 5.1 syntax to include its macro system, and allows programmers to define their own syntax. Metalua can provide syntactical support for several object-oriented styles, and can also provide syntax for turning simple type annotations into run-time assertions.

MoonScript [Cor11] is a programming language that supports class-based object-oriented programming. MoonScript compiles to idiomatic Lua code, but it does not perform compile-time type checking.

LuaInspect [Man11] is a tool that uses MetaLua to perform some code analysis. For instance, it flags unknown global variables and table fields, it checks the number of function arguments against signatures, and it infers function return values. However, it does not try to analyze object-oriented code and it does not perform compile-time type checking.

Tidal Lock [Fle13] is a prototype of another optional type system for Lua, which is written in Metalua. Tidal Lock covers a little subset of Lua. Statements include declaration of local variables, multiple assignment, function application, and the return statement. This means that Tidal Lock does not include any control-flow statement. Expressions include primitive literals, table indexing, function application, function declaration, and the table constructor, but they do not include binary operations.

A remarkable feature of Tidal Lock is the refinement of table types. This feature inspired us to also include it in Typed Lua, but in a simpler way and with different formalization.

PUC-Rio - Certificação Digital Nº 1012679/CA

The table type from Tidal Lock can only represent records, that is, it cannot describe hash tables and arrays yet, though we can refine them. Tidal Lock also includes field types to describe the type of the fields of a table type. The field types describe if a table field is mutable or immutable in a table type. Field types are the feature that allow the refinement of table types in Tidal Lock.

Tidal Lock is also a structural type system that relies on subtyping and local type inference. However, it does not support union types, recursive types, and variadic types. It also does not type any object-oriented idiom.

Sol [Ern13] is an experimental optional type system for Lua. Its type system is similar to ours, as it includes literal types, union types, and function types that handle variadic functions. However, it does not handle the refinement of tables and it includes different types for tables. Sol types tables as lists, maps, and objects. Its object types handle a specific object-oriented idiom that Sol introduces.

Lua Analyzer [Cla14] is an optional type system for Lua that is specially designed to work in the Löve Studio, an IDE for game developing using the Löve framework. It works in Lua 5.1 only, and uses type annotations inside comments. It is unsound by design because its dynamic type is both top and bottom in the subtyping relation.

Lua Analyzer shares some features with Typed Lua, and also has some interesting features that we do not have in Typed Lua. It has similar rules for handling the `or` idiom and discriminating union types inside conditions. However, these rules are limited to the `nil` tag only. It also includes different types for typing tables. It includes regular record types that maps names to types, array types, and map types. Even though it does not support the refinement of tables, it allows the definition of nominal table types that simulate classes. This system allows it to type check custom class systems, which are common in Lua. Function types also support multiple return values and variadic functions, but they do not support overloading the return type. Recently, it included experimental support for type aliases and generics.

Luacheck [Mel14] is a tool that performs static analysis on Lua code. It can flag access to undeclared globals and unused local variables, but it does not perform static type checking.

Ravi [Maj15] is an experimental Lua dialect. Ravi introduces optional static typing for Lua to improve run-time performance. To do that, Ravi extends the Lua Virtual Machine to include new operations that take into account static type information. Currently, Ravi extends the Lua Virtual Machine to support few types: `integer`, `number`, arrays of integers, and arrays

PUC-Rio - Certificação Digital Nº 1012679/CA

of numbers.

6.2 Other projects

Typed Racket [THF08] is a statically typed version of the Racket language, which is a Scheme dialect. The main purpose of Typed Racket is to allow programmers to combine untyped modules, which are written in Racket, with typed modules, which are written in Typed Racket. It also uses local type inference to deduce the type of unannotated expressions.

The main feature of Typed Racket's type system is *occurrence typing* [THF10]. It is a novel way to use type predicates in control flow statements to refine union types. Occurrence typing is not sound in the presence of mutation. As these kinds of checks are common in other languages, related systems have appeared [GSK11, Win11, Pea13].

The type system of Typed Racket also includes function types, recursive types, and structure types. Its function types also handle multiple return values, and there is also a way to describe function types that have optional arguments. Its structure types are similar to our interfaces, as they describe record types. The type system is also structural and based on subtyping. It also includes the dynamic type `Any`, which is the top type in the system. Typed Racket also supports polymorphic functions and data structures.

Typed Clojure [BS12] is an optional type system for Clojure. Although Clojure is a Lisp dialect that runs on the Java Virtual Machine, Common Language Runtime, and JavaScript, Typed Clojure runs only on the Java Virtual Machine. Perhaps, this restriction pushed Typed Clojure to support Java classes and some Java types such as `Long`, `Double`, and `String`. Typed Clojure also provides optional type annotations and uses local type inference to deduce the type of unannotated expressions. It also assigns the type `Any` to unannotated function parameters, which is the top type in the type system.

The type system of Typed Clojure includes polymorphic function types, union types, intersection types, lists, vectors, maps, sets, and recursive types. Function types can also have rest parameters, which are similar to our variadic types, but can only appear on the input parameter of function types. In fact, its function types cannot return multiple results. It also uses occurrence typing to allow control flow statements to refine union types. The type system is also structural and based on subtyping.

Dart [Goo11] is a new class-based object-oriented programming language. It includes optional type annotations and compiles to JavaScript. The type system of Dart is nominal and includes base types, function types, lists,

PUC-Rio - Certificação Digital Nº 1012679/CA

and maps. It also supports generics, and the programmer can define generic functions, lists, and maps. Unlike Typed Lua, Dart is unsound by design.

Even though Dart has optional typing and static types by default do not affect run-time semantics, it has an execution mode that affects run-time. The *checked mode* inserts run-time assertions that verifies whether static types match run-time tags. The *production mode* is the default execution mode that does not include any assertions.

TypeScript [Mic12] is a JavaScript extension that includes optional type annotations and class-based object-oriented programming. It also uses local type inference to deduce the type of unannotated expressions. The type system of TypeScript is structural, based on subtyping, and supports generics. It includes the dynamic type, primitive types, union types, function types, array types, tuple types, recursive types, and object types. Unlike Typed Lua, TypeScript uses arrays to represent variadic functions and multiple return values.

Even though TypeScript is unsound by design, Bierman et al. [BAT14] shows how to make TypeScript sound. They use a reduced core of TypeScript to formalize a sound type system for TypeScript, but also to formalize its current unsound type system.

TeJaS [LPGK13] is a framework for the construction of different type systems for JavaScript. The authors created a base type system for JavaScript with extensible typing rules that allow the experimentation of different static analysis. They used TeJaS to create a type system that simulates the type system of TypeScript.

Politz et al. [PGK12] proposes semantics and types for objects with first-class member names, a well-known feature from scripting languages. Their type system uses string patterns to describe the members of an object, and define a complex subtyping relation to validate these patterns. They also provide an implementation of their system to JavaScript.

Gradualtalk [ACF+13] is a Smalltalk dialect that supports gradual typing. The type system combines nominal and structural typing. It includes function types, union types, structural types, nominal types, a self type, and parametric polymorphism. The type system also relies on subtyping and consistent-subtyping.

Gradualtalk inserts run-time checks that ensure dynamically typed code does not violate statically typed code. Allende et al. [AFT13] perform a careful evaluation about cast insertion in Gradualtalk. They report that usually cast insertions impact on execution performance, so Gradualtalk also has an option that allows programmers to turn them off, downgrading Gradualtalk to an

PUC-Rio - Certificação Digital Nº 1012679/CA

optional type system.

Reticulated Python [VKSB14] is a Python compiler that supports gradual typing. The type system is structural and based on subtyping. It includes base types, the dynamic type, list types, dictionary types, tuple types, function types, set types, object types, class types, and recursive types. It includes class and object types to differentiate the type of class declarations and instances, respectively. It also uses local type inference. Besides static type checking, Reticulated Python also introduces three different approaches for inserting run-time assertions.

Mypy [Leh14a] is an optional type system for Python. The type system of mypy is similar to the type system of Reticulated Python, but mypy does not insert run-time checks and it has parametric polymorphism. In contrast, Reticulated Python can type variadic functions, but mypy cannot. Recently, Guido van Rossum, Python's author, proposed a standard syntax for type annotations in Python [vR14] that is extremely inspired by mypy [vRLL14]. The main goal of this proposal is to make easier building static analysis tools for Python. Typing [Leh14b] is a tool that is being developed to implement this proposal.

Hack [Fac14] is a new programming language that runs on the Hip Hop Virtual Machine (HHVM). The HHVM is a virtual machine that executes Hack and PHP programs. We can view Hack as an extension to PHP that combines static and dynamic typing. The type system of Hack includes generics, nullable types, collections, and function types.

The Ruby Type Checker [RTSF13] is a library that performs type checking during run-time. The library provides type annotations that the programmer can use on classes and methods. Its type system includes nominal types, union types, intersection types, method types, parametric polymorphism, and type casts.

Grace [BBH+13] is an object-oriented language with optional typing. Grace is not a dynamically typed language that has been extended with an optional type system, but a language that has been designed from scratch to have both static and dynamic typing. Homer et al. [HBNB13] explores some useful patterns that derive from Grace's use of objects as modules and its brand of optional structural typing, which can also be expressed with Typed Lua's modules as tables.

PUC-Rio - Certificação Digital Nº 1012679/CA