PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Felipe V. Côrtes

Type Extractor

Projeto Final de Programação

Especificação apresentada como requisito parcial para obtenção de grau da disciplina Projeto final de Programação em Informática, do Departamento de Informática da PUC-Rio .

Advisor: Prof. Roberto Ierusalimschy

Rio de Janeiro
March 2018

Abstract

Cortes,Felipe; Ierusalimschy, Roberto (Advisor). **Type Extractor**. Rio de Janeiro, 2018. 18p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The acquisition of triangular meshes typically introduces undesired noise...

Keywords

Geometry Processing; Mesh Denoising; Adaptive Patches.

Resumo

Cortes,Felipe; Ierusalimschy, Roberto. **Extrator de tipos**. Rio de Janeiro, 2018. 18p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A aquisição de malhas triangulares normalmente introduz ruídos indesejados...

Palavras-chave

Procesamento Geométrico; Remoção de ruído de malha; Vizinhança adaptativa.

Table of contents

List of figures

List of tables

List of algorithms

List of codes

List of Abreviations

ADI – Análise Digital de Imagens
BIF – *Banded Iron Formation*

1
Introduction

There are several reasons that motivate the adoption of statically typed languages. Maintaining large systems built with dynamic types can become a nightmare due to the lack of type information (TAKIKAWA et al.,). Typed languages also generally has better performance because compile-time type information helps generating optimized machine code. However, programmers are frequently left empty-handed when inspecting dynamically typed code while having to re-write systems to a statically typed languaged if gradually typed languages are not an option.

Inspired by the challenge of inspecting dynamically typed code, we built a type extractor for the Lua programming language. By inspecting a program's execution during runtime, it can generate enough information to help programmers visualize the types being transfered between functions of their program. The software output can be used as an useful documentation, while also helping programmers migrate code to a statically typed one or even for debugging.

The document is structured as follows. In Chapter 2 we present previous work related to type systems in Lua. In Chapter 3 we describe the software goal. Chapter 4 explain the software modules and how they interact. In Chapter 5 it's shown the software key functions, the modules relationship and basic utilization. In Chapter 6 we present and discuss some results obtained by the type extractor on some Lua benchmarks. Finally on Chapter 7 we present our conclusion and future work.

2
Previous Work

Typed Lua (MAIDL; MASCARENHAS; IERUSALIMSCHY, 2014) has already defined an optional type system for the language. More than enriching documentation, this extension ensures static type safety while preserving Lua idioms. Typed Lua encodes the main data structure mechanism from Lua into arrays, records, tuples and maps. It uses a bracket syntax to denote table types:

```
local t: { string: number } = { foo = 1 }
local x: number = t.foo -- x gets 1
```

The type system features are designed to be lightweight and type-safe and extends for typing object, classes and modules by adding few type annotations.

Lua Type System has also been explored for scripting optimization with Pallene (GUALANDI; IERUSALIMSCHY,). The language uses an optional type system for integrating seamlessly with Lua's dynamic code.

```
function add ( x : float , y : float ): float
    return x + y
end
```

As opposed to Typed Lua, Pallene is designed for efficiency. It performs runtime checks to ensure type safety with a tweak flexibility.

3
Project Scope

The extractor can analyse each function call and return by using reflection properties of the Lua programming language. With the debug library, we ca n register hook functions to inspect the behaviour of a program's execution and then compute the types of functions and variables present in the code .
The Type Extractor can be used by two approaches.

– Full Analysis: A full program analysis can be made by passing a Lua program as input to the extractor. In this approach, each possible fun ction call and return types will be analysed.
– Inspection library: An auxiliar library, capable of registring specific functions for inspection. In this approach, the programmer can sel ect what part of the program they want to analyse.

In the end of each execution, a report will be generated. This report has information about parameters and return types of each analysed function.
These usage scenarios enables the extractor to be used as an auxiliary tool for migrating from dynamically to statically typed languages. It also ser ves as a good documentation for functions parameter and return types. Giving tools for understanding the type relations inside a program helps progra mmers to debug and optimize dynamically typed code.

4
Project Specification

Equation example 1:

$$
\begin{aligned}
\min_{u} \int_{x_i \in X} \int_{x_j \in X} q_{ij} u_i u_j dada + \int_{x_i \in X} ||x' - x_i|| u_i da \\
s.t. \quad u \in [0,1] \quad \wedge \quad \int_{x_i \in X} uda = a_0,
\end{aligned}
\tag{4-1}
$$

Equation exmaple 2:

$$
\begin{aligned}
\min_{\mathbf{u}} \alpha \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} + \beta \mathbf{d}^T a' \mathbf{A} \mathbf{u} + \gamma \mathbf{u}^T \mathbf{G}^T \mathbf{G} \mathbf{u} + \delta \mathbf{f}^T a' \mathbf{A} \mathbf{u} \\
s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0.
\end{aligned}
\tag{4-2}
$$

Equation example 3:

$$
\mathbf{G} = (g_{ij}) = \begin{cases} \sum_{f_k \in N_f(f_i)} l_{ik} & i = j \\ -l_{ij} & e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}
\tag{4-3}
$$

Code 1: Mean Filter

```r
# --------------------------------------------------------------------#
# Create filter function
# l is the width of window
# --------------------------------------------------------------------#
meanfilter <- function( l, imagem ) {
  if( l%%2 == 0 )
    print("Please, type an odd number!")
  imagem.result <- imagem
  lp1d2 <- (l-1)/2
  L <- dim(imagem)[1]
  C <- dim(imagem)[2]
  for( j in as.integer(lp1d2+1) : as.integer(C-lp1d2)) {
    for( i in as.integer(lp1d2+1) : as.integer(L-lp1d2)) {
      imagem.result[i,j] <- mean(imagem[as.integer(i-lp1d2):as.integer(i+lp1d2), as.integer(j-lp1d2):as.integer(j+lp1d2)])
    }
  }
```

```
  print("Image filtred with success!")
  return(imagem.result)
}
# -------------------------------------------------------------------#
# End of Script.
# -------------------------------------------------------------------#
```

Algorithm 1: Escolha das amostras inicias

Input: Malha e quantidade de pontos a ser amostrado

Output: Pontos amostrados na malha

Crie um vetor de números randômicos entre $[0, 1]$ *com a quantidade de pontos a ser amostrada e ordene-o*

Calcule a área total dos triângulos da malha

for $i = 0$ **to** *numeroDePontos* **do**

  Navegue entre as faces acumulando a sua $\frac{area}{areaTotal}$ *até achar a face com valor acumulado* $\geqslant$ *numerosRandomicos[i]*

  Pegue um ponto randômico dentro da face utilizando o método de Turk e adicione no vetor do resultado

5
Development

Table example. Table 5.1 shows results.

Table 5.1: Results for devil mesh

	Mean Vertex Distance	L2 Vertex Based	Mean Quadric	MSAE	L2 Normal Based	Tangential	Mean Discrete Curvature	Area Error	Volume Error
(??)	0.061277	0.110973	0.236219	19.697900	0.055170	0.047678	0.090284	0.051443	0.045645
(??)	0.001293	0.002800	0.002289	21.237300	0.021589	0.013026	0.087991	0.000364	0.002621
(??)	0.001439	0.002880	0.003540	14.043200	0.012654	0.008911	0.055849	0.007806	0.000582
(??)	0.000713	0.001537	0.001824	12.171400	0.009654	0.005781	0.054567	0.005617	0.000425
(??)	0.002531	0.004560	0.007108	13.830100	0.017459	0.010314	0.114528	0.001686	0.001786
(??)	0.001623	0.003079	0.005048	10.454200	0.015233	0.008054	0.094668	0.002629	0.001326
(??)	0.000737	0.001548	0.001493	16.880800	0.014129	0.006974	0.079952	0.000209	0.002375
Ours	0.000987	0.001902	0.002686	11.574200	0.010632	0.006796	0.075106	0.003970	0.000722

5.1
Comparison

6
Results

We proposed an algorithm for triangular mesh denoising with detail preservation...

Code 2: Mean Filter

```
# -----------------------------------------------------------------------#
# Create filter function
# l is the width of window
# -----------------------------------------------------------------------#
meanfilter <- function( l, imagem ) {
  if( l%%2 == 0 )
    print("Please, type an odd number!")
  imagem.result <- imagem
  lp1d2 <- (l-1)/2
  L <- dim(imagem)[1]
  C <- dim(imagem)[2]
  for( j in as.integer(lp1d2+1) : as.integer(C-lp1d2)) {
    for( i in as.integer(lp1d2+1) : as.integer(L-lp1d2)) {
      imagem.result[i,j] <- mean(imagem[as.integer(i-lp1d2):as.integer(i+lp1d2), as.integer(j-lp1d2):as.integer(j+lp1d2)])
    }
  }
  print("Image filtred with success!")
  return(imagem.result)
}
# -----------------------------------------------------------------------#
# End of Script.
# -----------------------------------------------------------------------#
```

7
Final Considerations

We proposed an algorithm for triangular mesh denoising with detail preservation...

Code 3: Mean Filter

```
# ------------------------------------------------------------------------#
# Create filter function
# l is the width of window
# ------------------------------------------------------------------------#
meanfilter <- function( l, imagem ) {
  if( l%%2 == 0 )
    print("Please, type an odd number!")
  imagem.result <- imagem
  lp1d2 <- (l-1)/2
  L <- dim(imagem)[1]
  C <- dim(imagem)[2]
  for( j in as.integer(lp1d2+1) : as.integer(C-lp1d2)) {
    for( i in as.integer(lp1d2+1) : as.integer(L-lp1d2)) {
      imagem.result[i,j] <- mean(imagem[as.integer(i-lp1d2):as.
          integer(i+lp1d2), as.integer(j-lp1d2):as.integer(j+lp1d2)
          ])
    }
  }
  print("Image filtred with success!")
  return(imagem.result)
}
# ------------------------------------------------------------------------#
# End of Script.
# ------------------------------------------------------------------------#
```

8 Bibliography

GUALANDI, H. M.; IERUSALIMSCHY, R. Pallene: A companion language for lua. v. 189, p. 102393. ISSN 01676423. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0167642320300046>.

MAIDL, A. M.; MASCARENHAS, F.; IERUSALIMSCHY, R. Typed lua: An optional type system for lua. In: **Proceedings of the Workshop on Dynamic Languages and Applications**. New York, NY, USA: Association for Computing Machinery, 2014. (Dyla'14), p. 1–10. ISBN 9781450329163. Disponível em: <https://doi.org/10.1145/2617548.2617553>.

TAKIKAWA, A. et al. Is sound gradual typing dead? In: **Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**. Association for Computing Machinery. (POPL '16), p. 456–468. ISBN 978-1-4503-3549-2. Event-place: St. Petersburg, FL, USA. Disponível em: <https://doi.org/10.1145/2837614.2837630>.