Eclipse OMR Garbage Collection for Meta-tracing JIT-based Dynamic Languages

Joannah Nanjekye, David Bremner
Faculty of Computer Science, University of New Brunswick
Aleksandar Micic
IBM Canada
{jnanjeky, bremner} @unb.ca, aleksandar_micic@ca.ibm.com

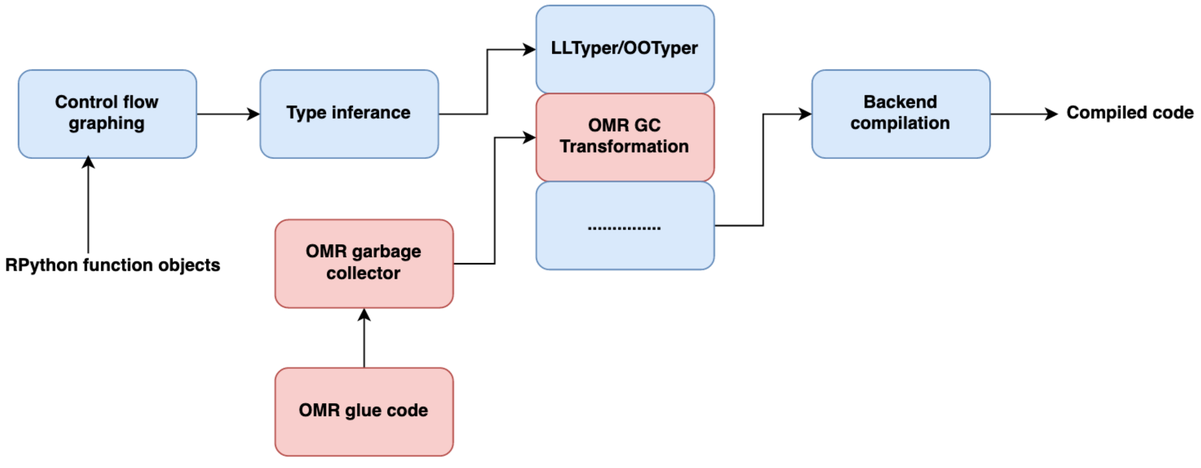
CAS–Atlantic

OVERVIEW

we present the design, implementation, and experimental evaluation of OMR-based garbage collection for tracing-JIT-based virtual machines that use the RPython translation toolchain.

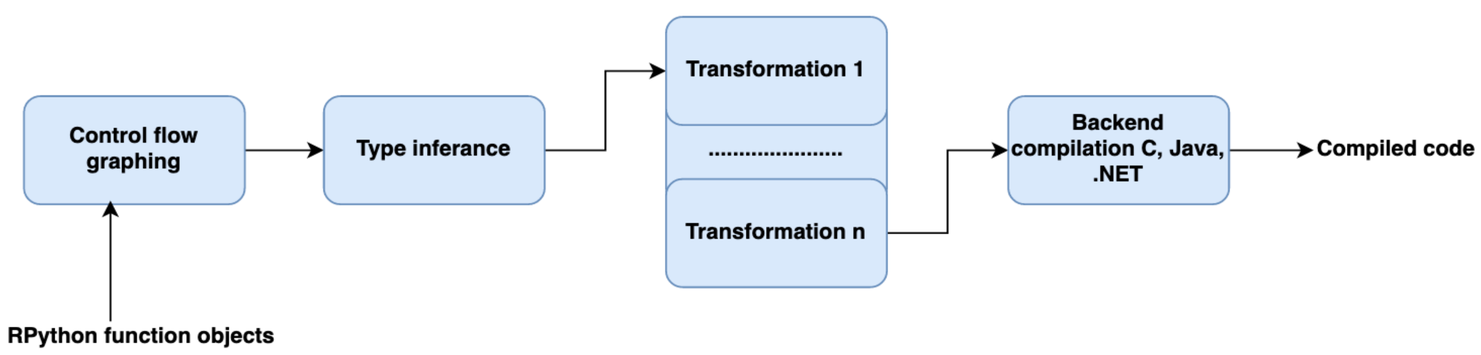
Our findings agree with existing work that using a GC framework results in a GC that is simple, easier to maintain, and has fewer bugs. The experimental results show that GC modularity through the Eclipse OMR GC framework can still achieve high performance for components like garbage collection for dynamic languages.

DESIGN: OMR GC IN THE RPYTHON FRAMEWORK



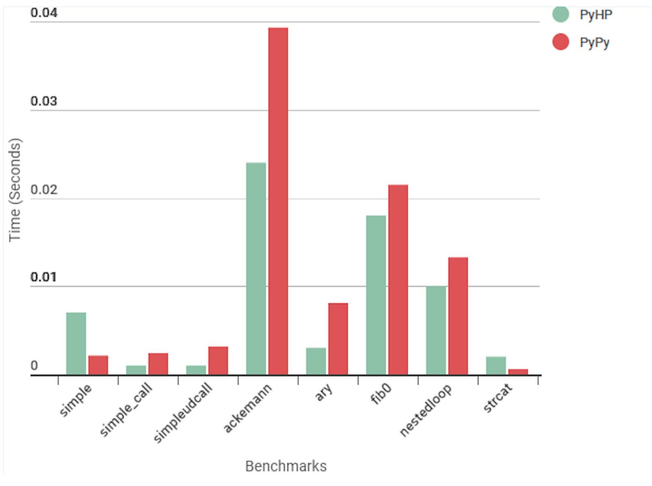
THE RPYTHON TRANSLATION TOOLCHAIN

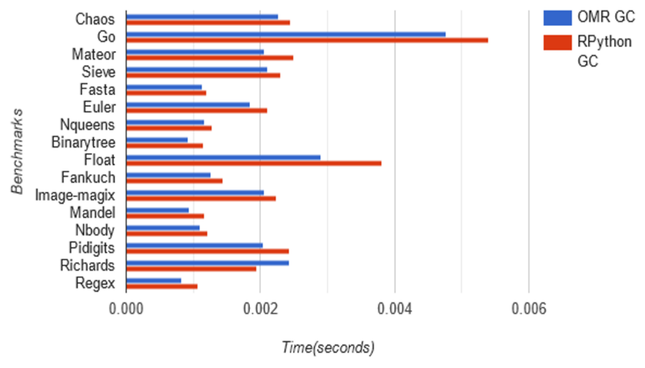
- Takes Python byte code
- The bytecode is converted to CFGS
- Type inference produces CFGs with typed variables
- Transformations convert the CFGs to a target environment like C, Java and .NET
- Garbage collection is equally achieved using a transformation

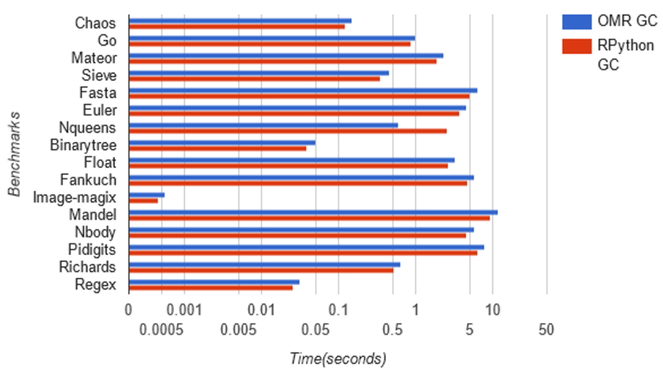


PERFORMANCE EVALUATION

- MI and CC is over 50% better
- OMR collection is 6%–20% faster
- OMR costs 20%–35% in traversal and allocation
- TLH and fast path optimizations able to reduce this overhead







SOFTWARE ENGINEERING EVALUATION

	OMR Garbage Collector	RPython Garbage Collector
Number of methods	22	26
Number of bytes	11.5kb	26kb
LOC	338	630
LLOC	224	538
SLOC	249	567
$\sum CC$	42	98
$\overline{CC}$	1.75	3.5

Raw code measurements

	OMR GC	Framework GC
Lack of cohesion of methods (LCOM)	2	1.5
Maintainability Index (MI)	31.28, A	14.12, B

$$MI = max\{0, 100\frac{171 - 5.2\ln V - 0.23G - 16.2\ln L + 50\sin\sqrt{2.4C}}{171}\}$$

Where:

(1) V = Halstead volume
(2) G = Total cyclomatic complexity
(3) L = SLOC
(4) C = Percentage of comment lines in radians