FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master's Thesis

Martin Holm Cservenka
Department of Computer Science
University of Copenhagen
djp595@alumni.ku.dk

Design and Implementation of Dynamic Memory Management in a Reversible Object-Oriented Programming Language

Supervisors: Robert Glück & Torben Ægidius Mogensen

Revision History

Revision	**Date**	**Author(s)**	**Description**
0.1	2017-04-30	Martin	ROOPL++ design and heap design discussion

Table of Contents

CHAPTER 1

Introduction

TODO

1.1 Reversible Computing

TODO

1.2 Reversible Object-Oriented Programming

The high-level reversible language Roopl (Reversible Object-Oriented Programming Language) was introduced in late 2016 and is, to our knowledge, the first of its kind [2]. The language extends the design of previously existing reversible imperative languages with object-oriented programming language features such as user-defined data types, class inheritance and subtype-polymorphism. As a first, Roopl successfully integrates the object-oriented programming (OOP) paradigms into the reversible computation setting using a static memory manager to maintain garbage-free computation, but at cost of "programmer usability" as objects only lives within `construct` / `deconstruct` blocks, which needs to be predefined, as the program call stacks is required to be reset before program termination.

1.3 Motivation

Roopl's block defined objects and lack of references are problematic when write complex, reversible programs using OOP methodologies. It has therefore been proposed to extend and partially redesign the language with dynamic memory management in mind, such that these shortcomings can be addressed, ultimately increasing the usability of reversible OOP. Work within the field of reversible computing related to heap manipulation [1], reference counting [3] and garbage collection [4] suggests that a Roopl extension is feasible.

1.4 Thesis Statement

TBD

1.5 Outline

TBC

CHAPTER 2

The Roolp++ Language

With the design and implementation of the Reversible Object-Oriented Programming Language (Roopl), the first steps into the uncharted lands of Object-Oriented Programming (OOP) and reversibility was taken. In this chapter, we will present Roopl++ , the natural successor of Roopl, improving the language's object instantiation by letting objects live outside `construct / deconstruct` blocks, allowing complex, reversible programs to be written using OOP methodologies. As with its predecessor, Roopl++ is purely reversible and each component of a program written in Roopl++ is locally invertible. This ensures no computation history is required nor added program size for backwards direction program execution.

`TODO: Arrays at some point`

`TODO: Roopl++ example program`

`Idea: Let Roopl++ be a superset of Roopl`

2.1 Syntax

A ROOPL++ program consists, analogously to a ROOPL program, of one or more class definitions, each with a varying number of fields and class methods.

ROOPL++ Grammar

$$
\begin{array}{rcll}
prog & ::= & cl^{+} & \text{(program)} \\
cl & ::= & \textbf{class}\ c\ (\textbf{inherits}\ c)^{?}\ (t\ x)^{*}\ m^{+} & \text{(class definition)} \\
t & ::= & \textbf{int} \mid c & \text{(data type)} \\
m & ::= & \textbf{method}\ q\texttt{(}t\ x\texttt{,}\ \ldots\texttt{,}\ t\ x\texttt{)}\ s & \text{(method)} \\
s & ::= & x\ \odot\texttt{=}\ e \mid x\ \texttt{<=>}\ x & \text{(assignment)} \\
 & \mid & \textbf{if}\ e\ \textbf{then}\ s\ \textbf{else}\ s\ \textbf{fi}\ e & \text{(conditional)} \\
 & \mid & \textbf{from}\ e\ \textbf{do}\ s\ \textbf{loop}\ s\ \textbf{until}\ e & \text{(loop)} \\
 & \mid & \textbf{new}\ c\ x & \text{(object construction)} \\
 & \mid & \textbf{delete}\ x & \text{(object deconstruction)} \\
 & \mid & \textbf{call}\ q\texttt{(}x\texttt{,}\ \ldots\texttt{,}\ x\texttt{)} \mid \textbf{uncall}\ q\texttt{(}x\texttt{,}\ \ldots\texttt{,}\ x\texttt{)} & \text{(local method invocation)} \\
 & \mid & \textbf{call}\ x\texttt{::}q\texttt{(}x\texttt{,}\ \ldots\texttt{,}\ x\texttt{)} \mid \textbf{uncall}\ x\texttt{::}q\texttt{(}x\texttt{,}\ \ldots\texttt{,}\ x\texttt{)} & \text{(method invocation)} \\
 & \mid & \textbf{skip} \mid s\ s & \text{(statement sequence)} \\
e & ::= & \overline{n} \mid x \mid \texttt{nil} \mid e\ \otimes\ e & \text{(expression)} \\
\odot & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{\^{}} & \text{(operator)} \\
\otimes & ::= & \odot \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{\&} \mid \texttt{|} \mid \texttt{\&\&} \mid \texttt{||} \mid \texttt{<} \mid \texttt{>} \mid \texttt{=} \mid \texttt{!=} \mid \texttt{<=} \mid \texttt{>=} & \text{(operator)}
\end{array}
$$

Syntax Domains

$$
\begin{array}{lll}
prog \in \text{Programs} & s \in \text{Statements} & n \in \text{Constants} \\
cl \in \text{Classes} & e \in \text{Expressions} & x \in \text{VarIDs} \\
t \in \text{Types} & \odot \in \text{ModOps} & q \in \text{MethodIDs} \\
m \in \text{Methods} & \otimes \in \text{Operators} & c \in \text{ClassIDs}
\end{array}
$$

Figure 2.1: Syntax domains and EBNF grammar for ROOPL++

2.2 Object Instantiation

TODO

CHAPTER 3

Compilation

3.1 Dynamic Memory Management

Native support of complex data structures is not a trivial matter to implement in a reversible computing environment. Variable-sized records and frames needs to be stored efficiently in a structured heap, while avoiding garbage build-up to maintain reversibility. A reversible heap manager layout has been proposed for a simplified version of the reversible functional language RFun and later expanded to allow references to avoid deep copying values [1, 5, 3].

The following section presents a discussion of various heap manager layouts along with their advantages and disadvantages in terms of implementation difficulty, garbage build-up and the OOP domain.

3.1.1 Heap manager layout

Fixed-sized records

One heap per record

Variable-sized records

Small number of fixed-sized blocks

Shared heap, record-specific free lists

References

[1] Axelsen, H. B. and Glück, R. “Reversible Representation and Manipulation of Constructor Terms in the Heap”. In: *RC ’13: Proceedings of the 5th International Conference on Reversible Computation* (2013), pp. 96–109.

[2] Haulund, T. “Design and Implementation of a Reversible Object-Oriented Programming Language”. Master’s Thesis. University of Copenhagen, DIKU, 2016.

[3] Mogensen, T. Æ. “Reference Counting for Reversible Languages”. In: *RC ’14: Proceedings of the 6th International Conference on Reversible Computation* (2014), pp. 82–94.

[4] Mogensen, T. Æ. “Garbage Collection for Reversible Functional Languages”. In: *RC ’15: Proceedings of the 7th International Conference on Reversible Computation* (2015), pp. 79–94.

[5] Yokoyama, T., Axelsen, H. B., and Glück, R. “Towards a Reversible Functional Language”. In: *RC 11: Proceedings of the Third International Conference on Reversible Computation* (2011), pp. 14–29.