発表概要

ごみ集めにおける 自動ルート保護と手動ルート保護の併用手法

半澤 順一[1,a)]　小宮 常康[1]

2020年6月2日発表

ごみ集め（GC）における GC ルートを保護する方式として，手動ルート保護と自動ルート保護が存在する．手動ルート保護を採用する場合は，処理系の開発者自身が処理系コードにルート保護用コードを挿入する必要があり，処理系コードが煩雑となるが，メモリリークは起きにくい．ただし，必要箇所にルート保護用コードを挿入し忘れると予測不能なバグが処理系に生じるため，細心の注意を払う必要がある．一方，保守的 GC に代表される自動ルート保護を採用する場合は，開発者が処理系コードにルート保護用コードを挿入する必要はなく，処理系コードがシンプルになるが，過剰なルートスキャンによってメモリリークを起こす恐れがある．従来の処理系開発では，どちらか一方のルート保護方式を選択しなければならない．しかし，両保護方式を併用できれば，各々の特長を活かした処理系開発や，1 つの処理系の中でルート保護方式を適切に使い分けることができるようになる．たとえば，開発初期は GC の確実な動作を実現するために自動ルート保護を採用し，後に GC 性能を追求して手動ルート保護に移行したり，あるいは，外部コードのみ自動ルート保護とすることが考えられる．本研究では，関数単位でどちらの保護を行うか指定できるようにすることで，手動ルート保護と自動ルート保護の共存を実現した．自動ルート保護の関数には，開発した変換器によってルート保護用コードが自動的に挿入される．本発表では，提案手法を述べ，GNU Common Lisp と SIOD 処理系のソースコードを変換したときの性能評価を報告する．

Presentation Abstract

A Method for Employing Both Manual Root Protection and Automatic Root Protection in Garbage Collector

Junichi Hanzawa[1,a)]　Tsuneyasu Komiya[1]

Presented: June 2, 2020

There are two garbage collection (GC) root protection styles: *manual root protection* and *automatic root protection*. If a programming language is implemented with the manual root protection style, the implementor has to add code for protecting GC roots explicitly. Since the protection code is scattered around the implementation code, the implementation becomes somewhat complicated, but memory leaks rarely occur, though it may cause unpredictable bugs if the implementor forgets to insert protection code. On the other hand, if the automatic root protection style, which is typically employed by conservative GC, is used, the implementor does not need to insert protection code, but memory leaks may occur due to excessive root scan. Typically, one of these styles of root protection must be chosen prior to implementing a programming language. However, if both protections can be used together, it allows taking advantage of both styles. For example, even if you adopt automatic root protection at an early stage of implementation in order to achieve reliable behavior of the GC, you can shift to manual root protection gradually in order to improve GC performance, or use automatic root protection only for external code. In this research, we propose a method of allowing the coexistence of manual root protection and automatic root protection. In our proposal, annotation of functions is used for choosing protection styles. Root protection code in functions annotated with *automatic root protection* is inserted by our C-to-C translator. This presentation shows the details of our proposal and reports the performance evaluation on GNU Common Lisp and SIOD.

This is the abstract of an unrefereed presentation, and it should not preclude subsequent publication.

[1] 電気通信大学大学院情報理工学研究科
Graduate School of Informatics and Engineering, The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] j.hanzawa@uec.ac.jp