情報処理学会論文誌　プログラミング　Vol. 4　No. 3　97 (June 2011)

発表概要

例外処理と2返戻値法を用いた効率的な継続の実現法

白畑有加[†1]　前田敦司[†1]　山口喜教[†1]

プログラムの処理の流れを扱う概念に，継続がある．継続とは，ある処理のある瞬間における,「次にどのような処理を行うのか」という処理過程の未来全体を表すものである．Scheme では継続をファーストクラスの機能としてサポートしており，call/cc 関数を用いて継続を扱うことができる．しかし，そのほかほとんどのプログラミング言語は継続を明示的に扱っていない．そのような言語に継続をファーストクラスの機能とし導入する手法として CPS 変換や例外処理を用いた手法など様々な研究がなされているが，導入後の実行効率が大幅に落ちてしまうという問題点をかかえている．本発表では対象言語を JavaScript とし，Pettyjohn らの手法をもとにした新しい，継続を導入する変換手法を提案する．変換後の実行効率の向上のため，継続から処理を再開するときのコードを別途定義する．これにより継続を使用しない場合の処理は変換前後で実行速度が変わらない．また，継続による処理の再開位置を示す Resumption Point を導入することで変換後のコードの重複を削減した．さらに，本発表では継続をキャプチャするタイミングに例外処理を用いる手法と 2 返戻値法の両方を使い分ける．このハイブリッドな継続のキャプチャ手法の採用により，さらに変換後の実行効率の向上を見込むことができる．

Efficient Implementation of Continuations with Exception Handling and Two Return Values Method

Yuka Shirahata,[†1] Atusi Maeda[†1]
and Yoshinori Yamaguchi[†1]

Continuation is an idea which deals with flow of control inprograms. Although Scheme has continuation as a first-class feature, most other programming languages don't support it explicitly. Various approaches have proposed to adopt the continuation as a first-class feature in such languages, e.g., CPS conversion and stack-traversal and rebuilding using exception. But these conversions have significant impact on execution efficiency and slows down the execution of programs after conversion considerably. In this presentation, we propose a new transformation algorithm based on Pettyjohn, et al.'s approach, which can introduce continuation more efficiently to programming languages that don't support them (we use JavaScript as a specific example). To improve execution efficiency after transformation, we separately generate the code run in ordinary execution and the code executed when continuation is invoked. When executed without using continuations, the execution efficiency after transformation is almost equal to the one before transformation. Since we separately generate the code for two cases, code size may increase after conversions. This code expansion is reduced by sharing code using labels called Resumption Point, which marks the point to resume when continuations are invoked. We use two techniques for stack traversal when capturing continuations: exception handling and two return values method. We combine these two techniques to get better performance in continuation capturing, and believe our approach has the advantages over existing techniques.

（平成 23 年 1 月 21 日発表）

†1 筑波大学大学院システム情報工学研究科
Graduate School of Information Engineering, University of Tsukuba

© 2011 Information Processing Society of Japan