CPS計算の線形型による拡張

中村光希[*1]　関山太朗[*2]　五十嵐淳[*1]

*1：京都大学　*2：国立情報学研究所

背景: CPS計算 [Thielecke,1997]

- CPS の項のみを記述できる計算体系
- IR に適した言語
  - 関数呼び出しがジャンプとして扱える
  - 自然にコントロールオペレータを扱える
- IR の理論研究基盤 [Torrens et al, 2024]
  - CPS計算の簡約規則が IR 上の最適化に相当

目的: IR の検証・探求

- リソース管理に関するコンパイラの最適化の安全性の検証に繋げる
- 線形型により IR 上でリソース管理の最適化を行い、表層言語コンパイラの最適化の負担を減らす
- 長期目標: 検証しやすい実用的な汎用 IR を作る
  - well-verified LLVM IR のようなもの

- λ計算からの CPS変換の対象言語
- 関数呼び出しと let 束縛の構文
- IR 上の最適化機能:
  - 未使用のコードの削除
  - 関数のインライン化

本研究

CPS計算 + 線形型

値の使用回数(use, $\kappa ::= 1 \mid \omega$)を数える型
1...ちょうど1回, ω...任意回

拡張の概要

- 型と構文における線形型の use の情報の保持
- CPS 変換時に導入される継続に use 1 をつける

貢献

- 追加の IR の最適化機能: use の情報を用いた、不要な束縛の削除

本体系: CPS計算+線形型

構文　use の情報を保持

型 $\tau ::= \neg^{\kappa} \vec{\tau}$　(関数型)

$\vec{\tau} \to^{\kappa} \bot$ のこと

式 $e ::= x_1(\vec{x_2})$　(関数呼び出し)

$\mid e_1\{x_1^{\kappa} = (\vec{x_2}).e_2\}$　(let 束縛)

- let $x_1^{\kappa} = \lambda \vec{x_2}.e_2$ in e_1 のこと
- 構文でも use の情報を保持

use を活用

簡約規則

Jump reduction（関数呼び出しの簡約規則）

$C[k(\vec{x})]\ \{k^{1} = (\vec{y}).b\} \rightsquigarrow_j C[b[\vec{x}/\vec{y}]]$

use 1 の値を使用後に削除

use による拡張

CPS変換　λ計算(線形性なし)からの変換

$[\![\lambda x.e]\!] = k(f)\{f^{\omega} = (x, k).[\![e]\!]\}$

継続の use は 1

$[\![e_1\ e_2]\!] = [\![e_1]\!]\{k^{1} = (y_1).[\![e_2]\!]\{k^{1} = (y_2).y_1(y_2, k)\}\}$

右辺に use を付加

証明済みの本体系の性質

✅ 型システムの健全性

項が well-typed なら、実行時エラーが起こらない

✅ CPS変換の型保存性

CPS変換前の項が well-typed なら、CPS変換後の項も well-typed

今後示すべき本体系の性質

- □ 合流性があること
- □ Jump reduction の前後の項が contextual equivalent であること
  - □ 併せて contextual equivalence の定式化

さらなる拡張

- □ 多相型への対応
  - 型保存 CPS 変換 [Sekiyama and Tsukada, 2021]との橋渡しにもなる
- □ 参照型への対応