コード生成言語に限定継続 shift0/reset0 を加えた型システムの設計

大石純平

筑波大学 大学院
プログラム論理研究室

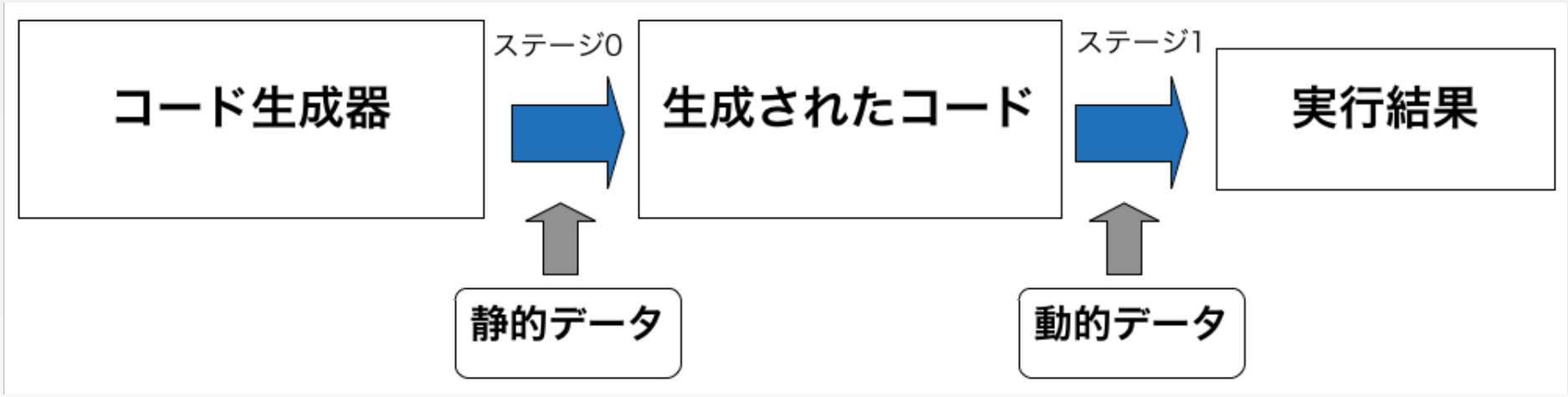
2016/7/12

アウトライン

アウトライン

アウトライン

段階的計算 (Staged Computation)



- コード生成ステージとコード実行ステージ
- 「保守性・再利用性の高さ」と「実行性能の高さ」の両立

⇒ 段階的計算をサポートするプログラム言語

アウトライン

コード生成における課題

生成されたコードの信頼性 (正しさ)

- パラメータに応じて，非常に多数のコードが生成される
- 生成したコードのデバッグが容易ではない

コード生成における課題

生成されたコードの信頼性 (正しさ)

- パラメータに応じて，非常に多数のコードが生成される
- 生成したコードのデバッグが容易ではない

従来研究

- コード生成プログラムが，安全なコードのみを生成する事を保証
- let 挿入等を実現する**計算エフェクトを含む場合の安全性保証は研究途上**

アウトライン

shift0 / reset0

アウトライン

例 1: 行列の積

$n \times n$ **行列の積**: `C = A B`

```
for i = 1 to n
  for k = 1 to n
    for j = 1 to n
      c[i][j] += a[i][k] * b[k][j]
```

例 1: 行列の積

$n \times n$ **行列の積**: C = A B

```
for i = 1 to n
  for k = 1 to n
    for j = 1 to n
      c[i][j] += a[i][k] * b[k][j]
```

ループの展開:

```
for i = 1 to n
  for k = 1 to n
    for j = 1 to n step 4
      c[i][j  ] += a[i][k] * b[k][j  ]
      c[i][j+1] += a[i][k] * b[k][j+1]
      c[i][j+2] += a[i][k] * b[k][j+2]
      c[i][j+3] += a[i][k] * b[k][j+3]
```

共通項のくくり出し

```
for i = 1 to n
  for k = 1 to n
    for j = 1 to n step 4
      c[i][j  ] += a[i][k] * b[k][j  ]
      c[i][j+1] += a[i][k] * b[k][j+1]
      c[i][j+2] += a[i][k] * b[k][j+2]
      c[i][j+3] += a[i][k] * b[k][j+3]
```

共通項のくくり出し

```
for i = 1 to n
  for k = 1 to n
    for j = 1 to n step 4
      c[i][j  ] += a[i][k] * b[k][j  ]
      c[i][j+1] += a[i][k] * b[k][j+1]
      c[i][j+2] += a[i][k] * b[k][j+2]
      c[i][j+3] += a[i][k] * b[k][j+3]
```

```
for i = 1 to n
  for k = 1 to n
    let t = a[i][k] in
    for j = 1 to n step 4
      c[i][j  ] += t * b[k][j  ]
      c[i][j+1] += t * b[k][j+1]
      c[i][j+2] += t * b[k][j+2]
      c[i][j+3] += t * b[k][j+3]
```

let 挿入 [Danvy 1996]

let 挿入

```
for i = 1 to n
  for k = 1 to n
    let t = a[i][k] in
    for j = 1 to n step 4
      c[i][j  ] += t * b[k][j  ]
      c[i][j+1] += t * b[k][j+1]
      c[i][j+2] += t * b[k][j+2]
      c[i][j+3] += t * b[k][j+3]
```

let 挿入 [Danvy 1996]

誤りのある let 挿入

```
for i = 1 to n
  let t = a[i][k] in          --- k is not bound
  for k = 1 to n in
    for j = 1 to n step 4
      c[i][j  ] += t * b[k][j  ]
      c[i][j+1] += t * b[k][j+1]
      c[i][j+2] += t * b[k][j+2]
      c[i][j+3] += t * b[k][j+3]
```

アウトライン

安全性の静的保証

```
C = A B
```

⇓：コード生成 (コード変換，最適化を含む)

```
for i = 1 to n
  for k = 1 to n
    let t = a[i][k]
    for j = 1 to n step 4
      c[i][j  ] += t * b[k][j]
      c[i][j+1] += t * b[k][j+1]
      c[i][j+2] += t * b[k][j+2]
      c[i][j+3] += t * b[k][j+3]
```

安全性の静的保証

動的に生成されたコードのデバッグは困難
⇒ コード生成の前に安全性を保証したい

アウトライン

研究の目的

表現力と安全性を兼ね備えたコード生成言語の構築

- 表現力: 多段階 let 挿入，メモ化等の技法を表現
- 安全性: 生成されるコードの一定の性質を静的に検査

研究の目的

表現力と安全性を兼ね備えたコード生成言語の構築

- 表現力: 多段階 let 挿入，メモ化等の技法を表現
- 安全性: 生成されるコードの一定の性質を静的に検査

本研究: より簡潔でより強力なコントロールオペレータに基づくコード生成体系の構築

- コントロールオペレータ shift0/reset0 を利用
- 種々のコード生成技法を表現
- 型システムを構築して型安全性を保証

アウトライン

アウトライン

目的

- shift0/reset0 を持つコード生成言語の型システムの設計
- 深く入れ子になった内側からの let 挿入 (多段階 let 挿入) など，関数プログラミング的実現

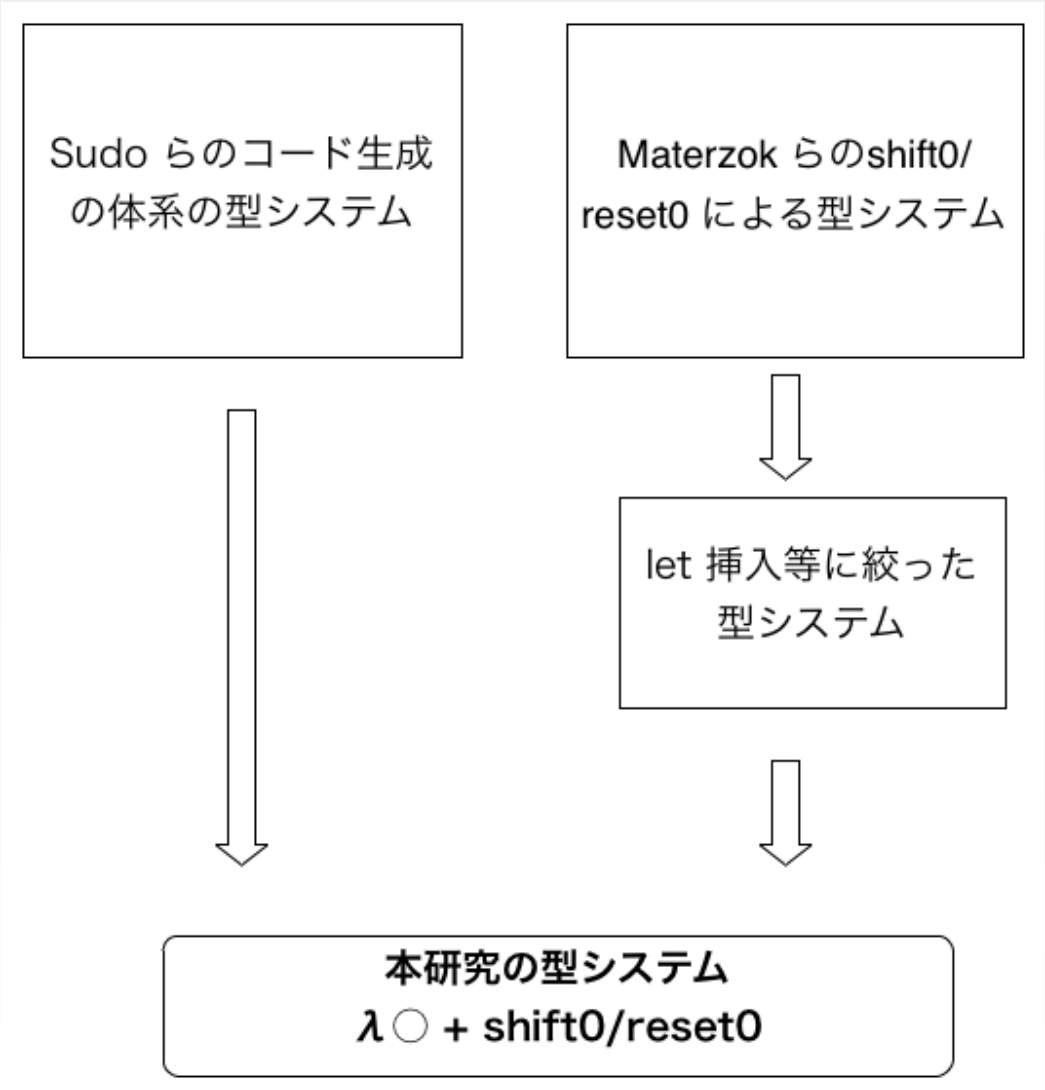
アウトライン

困難・問題点

- shift0/reset0 は shift/reset より強力であるため，型システムが非常に複雑
- コード生成言語の型システムも一定の複雑さ
- ⇒ 単純な融合は困難

アウトライン

本研究の手法


Sudo らのコード生成
の体系の型システム
Materzok らのshift0/
reset0 による型システム
let 挿入等に絞った
型システム
本研究の型システム
λ○ + shift0/reset0

アウトライン

型システムの設計方針

Typing rule

$$t ::= \mathrm{BasicType} \mid t \to t \mid \langle t \rangle^{\gamma}$$

Typing rule for code-level lambda:

$$\frac{\Gamma,\ \gamma_1 \geq \gamma,\ x : \langle t_1 \rangle^{\gamma_1} \vdash e\ :\ \langle t_2 \rangle^{\gamma_1}}{\Gamma \vdash \underline{\lambda} x.e\ :\ \langle t_1 \to t_2 \rangle^{\gamma}}\ (\gamma_1 \text{ is eigen var})$$

Typing rule for code-level let (derived rule):

$$\frac{\Gamma \vdash e_1\ :\ \langle t_1 \rangle^{\gamma} \quad \Gamma,\ \gamma_1 \geq \gamma,\ x : \langle t_1 \rangle^{\gamma_1} \vdash e_2\ :\ \langle t_2 \rangle^{\gamma_1}}{\Gamma \vdash \underline{\textbf{clet}}\ x = e_1\ \underline{\textbf{in}}\ e_2\ :\ \langle t_2 \rangle^{\gamma}}\ (\gamma_1 \text{ is eigen var})$$

Typing rule for code-level reset0:

$$\frac{\Gamma \vdash e\ :\ \langle t \rangle^{\gamma}}{\Gamma \vdash \underline{\textbf{reset0}}\ e\ :\ \langle t \rangle^{\gamma}}$$

Typing rule for code-level shift0:

$$\frac{\Gamma,\ k : (\langle t_1 \rangle^{\gamma_1} \Rightarrow \langle t_0 \rangle^{\gamma_0}) \vdash e\ :\ \langle t_0 \rangle^{\gamma_0}}{}$$

アウトライン

型が付く例 / 付かない例

$$
\begin{aligned}
e_1 = \ & \underline{\textbf{reset0}}\ \ \underline{\textbf{clet}}\ x_1 = \%3\ \underline{\textbf{in}} \\
& \underline{\textbf{reset0}}\ \ \underline{\textbf{clet}}\ x_2 = \%5\ \underline{\textbf{in}} \\
& \underline{\textbf{shift0}}\ k\ \rightarrow\ \underline{\textbf{clet}}\ y = t\ \underline{\textbf{in}} \\
& \underline{\textbf{throw}}\ k\ (x_1\ \underline{+}\ x_2\ \underline{+}\ y)
\end{aligned}
$$

If $t = \%7$ or $t = x_1$, then e_1 is typable.
If $t = x_2$, then e_1 is not typable.

$$
\begin{aligned}
e_2 = \ & \underline{\textbf{reset0}}\ \ \underline{\textbf{clet}}\ x_1 = \%3\ \underline{\textbf{in}} \\
& \underline{\textbf{reset0}}\ \ \underline{\textbf{clet}}\ x_2 = \%5\ \underline{\textbf{in}} \\
& \underline{\textbf{shift0}}\ k_2\ \rightarrow\ \underline{\textbf{shift0}}\ k_1\ \rightarrow\ \underline{\textbf{clet}}\ y = t\ \underline{\textbf{in}} \\
& \underline{\textbf{throw}}\ k_1\ (\underline{\textbf{throw}}\ k_2\ (x_1\ \underline{+}\ x_2\ \underline{+}\ y))
\end{aligned}
$$

If $t = \%7$, then e_1 is typable.
If $t = x_2$ or $t = x_1$, then e_1 is not typable.

アウトライン

まとめと今後

- **コードの型システムに** shift0 reset0 **を組み込んだ 型システムの設計を行った**
- **今後** answer type modification **に対応した型システムを設計し，**(subject reduction **等の**) **健全性の証明を行う**