多段階 let 挿入を行うコード生成言語の型システムの設計

大石純平
亀山幸義

筑波大学 大学院 コンピュータ・サイエンス専攻

2016/9/9

アウトライン

アウトライン

概要

プログラムを生成するプログラミング言語 (=**コード生成言語**) の安全性を保証する研究

概要

プログラムを生成するプログラミング言語 (=**コード生成言語**) の安全性を保証する研究

- 効率的なコードの生成
- 安全性の保証

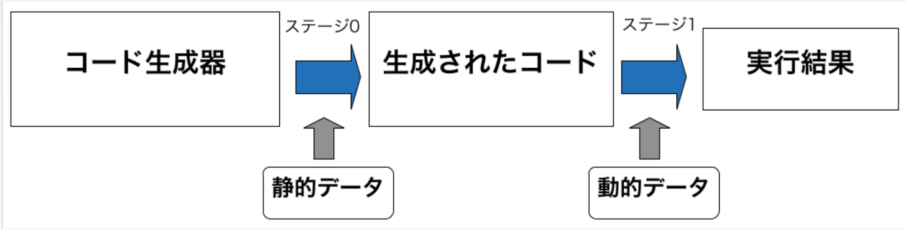
概要

プログラムを生成するプログラミング言語 (=コード生成言語) の安全性を保証する研究

- 効率的なコードの生成
- 安全性の保証

⇒ 多段階 let 挿入を効率的かつ安全に扱うための型システムを構築

コード生成（段階的計算）



- コード生成段階とコード実行段階

⇒ 段階的計算をサポートするプログラム言語 ⇒ コード生成言語

我々のコード生成言語による記述例

コード生成器 から 生成されるコードの例をみる

我々のコード生成言語による記述例

コード生成器 から 生成されるコードの例をみる

コード生成器 $\rightarrow$ 生成されるコード

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\textbf{コード生成器} \rightarrow \textbf{生成されるコード}$$

$(\underline{\textbf{int}}\ 3)$

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\text{コード生成器} \rightarrow \text{生成されるコード}$$

$$(\underline{\textbf{int}}\ 3) \rightarrow \texttt{<}4\texttt{>}$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコードの例をみる

$$\textbf{コード生成器} \to \textbf{生成されるコード}$$

$$(\underline{\textbf{int}}\ 3) \to <4>$$

$$(\underline{\textbf{int}}\ 3)\ \underline{+}\ (\underline{\textbf{int}}\ 5)$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコードの例をみる

$$\textbf{コード生成器} \to \textbf{生成されるコード}$$

$$(\underline{\textbf{int}}\ 3) \to \text{<}4\text{>}$$

$$(\underline{\textbf{int}}\ 3)\ \underline{+}\ (\underline{\textbf{int}}\ 5) \leadsto^* \text{<}3+5\text{>}$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\textbf{コード生成器} \to \textbf{生成されるコード}$$

$$(\underline{\textsf{int}}\ 3) \to <4>$$

$$(\underline{\textsf{int}}\ 3)\ \underline{+}\ (\underline{\textsf{int}}\ 5) \leadsto^{*} <3+5>$$

$$\underline{\lambda} x.x\ \underline{+}\ (\underline{\textsf{int}}\ 3)$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\text{コード生成器} \to \text{生成されるコード}$$

$$(\underline{\textbf{int}}\ 3) \to \texttt{<}4\texttt{>}$$

$$(\underline{\textbf{int}}\ 3)\ \underline{+}\ (\underline{\textbf{int}}\ 5) \rightsquigarrow^* \texttt{<}3+5\texttt{>}$$

$$\underline{\lambda} x.x\ \underline{+}\ (\underline{\textbf{int}}\ 3) \rightsquigarrow^* \texttt{<}\lambda x'.x' + 3\texttt{>}$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\textbf{コード生成器} \to \textbf{生成されるコード}$$

$$(\underline{\textbf{int}}\ 3) \to \texttt{<}4\texttt{>}$$

$$(\underline{\textbf{int}}\ 3)\ \underline{+}\ (\underline{\textbf{int}}\ 5) \rightsquigarrow^{*} \texttt{<}3+5\texttt{>}$$

$$\underline{\lambda} x.x\ \underline{+}\ (\underline{\textbf{int}}\ 3) \rightsquigarrow^{*} \texttt{<}\lambda x'.x' + 3\texttt{>}$$

$$\underline{\textbf{for}}\ x = n\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}\ e$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコード の例をみる

$$\text{コード生成器} \to \text{生成されるコード}$$

$$(\underline{\mathbf{int}}\ 3) \to <4>$$

$$(\underline{\mathbf{int}}\ 3)\ \underline{+}\ (\underline{\mathbf{int}}\ 5) \leadsto^{*} <3+5>$$

$$\underline{\lambda} x.x\ \underline{+}\ (\underline{\mathbf{int}}\ 3) \leadsto^{*} <\lambda x'.x' + 3>$$

$$\underline{\mathbf{for}}\ x = n\ \underline{\mathbf{to}}\ m\ \underline{\mathbf{do}}\ e \to <\mathbf{for}\ x = n\ \mathbf{to}\ m\ \mathbf{do}\ e>$$

我々のコード生成言語による記述例

コード生成器 から 生成されるコードの例をみる

$$\text{コード生成器} \rightarrow \text{生成されるコード}$$

$$(\underline{\textsf{int}}\ 3) \rightarrow <4>$$

$$(\underline{\textsf{int}}\ 3)\ \underline{+}\ (\underline{\textsf{int}}\ 5) \leadsto^{*} <3+5>$$

$$\underline{\lambda} x.x\ \underline{+}\ (\underline{\textsf{int}}\ 3) \leadsto^{*} <\lambda x'.x' + 3>$$

$$\underline{\textsf{for}}\ x = n\ \underline{\textsf{to}}\ m\ \underline{\textsf{do}}\ e \rightarrow <\textsf{for}\ x = n\ \textsf{to}\ m\ \textsf{do}\ e>$$

コードコンビネータ

- アンダーバーのついた演算子
- レベル 0 の項をコード化する
  ex. 1 : レベル 0 項 , $<1>$: コード化した項

power 関数のコード化

普通の power 関数

$$\begin{aligned}\text{power} = \;& \lambda x.\textbf{fix}\ \lambda f.\lambda n.\\ & \textbf{if}\ n = 0\ \textbf{then}\ 1\\ & \textbf{else}\ x\ \times\ (f\ (n-1))\end{aligned}$$

power 関数のコード化

コード化を行った power 関数 (power コード生成器)

$$\begin{aligned} \text{gen\_power} = \ & \underline{\lambda} x.\textbf{fix}\ \lambda f.\lambda n. \\ & \textbf{if}\ n = 0\ \textbf{then}\ (\underline{\textbf{int}}\ 1) \\ & \textbf{else}\ x\ \underline{\times}\ (f\ (n-1)) \end{aligned}$$

power 関数のコード化

コード化を行った power **関数** (power **コード生成器**)

$$\begin{aligned}\text{gen\_power} = \ & \underline{\lambda} x.\textbf{fix}\ \lambda f.\lambda n. \\ & \textbf{if}\ n = 0\ \textbf{then}\ (\underline{\textbf{int}}\ 1) \\ & \textbf{else}\ x \mathbin{\underline{\times}} (f\ (n-1))\end{aligned}$$

$n = 5$ **に特化したコードの生成を行う**

power 関数のコード化

コード化を行った power 関数 (power コード生成器)

$$\begin{aligned}\text{gen\_power} = \ & \underline{\lambda} x.\textbf{fix}\ \lambda f.\lambda n. \\ & \textbf{if}\ n = 0\ \textbf{then}\ (\underline{\textbf{int}}\ 1) \\ & \textbf{else}\ x \ \underline{\times}\ (f\ (n-1))\end{aligned}$$

$n = 5$ **に特化したコードの生成を行う**

$$\underline{\lambda} x.\text{gen\_power}\ x\ 5 \leadsto^{*} <\lambda x'.x' * x' * x' * x' * x' * 1>$$

power 関数のコード化

コード化を行った power 関数 (power コード生成器)

$$\begin{aligned}\text{gen\_power} = \ & \underline{\lambda} x.\textbf{fix}\ \lambda f.\lambda n. \\ & \textbf{if}\ n = 0\ \textbf{then}\ (\underline{\textbf{int}}\ 1) \\ & \textbf{else}\ x \ \underline{\times}\ (f\ (n-1))\end{aligned}$$

$n = 5$ に特化したコードの生成を行う

$$\underline{\lambda} x.\text{gen\_power}\ x\ 5 \rightsquigarrow^{*} \texttt{<}\lambda x'.x' * x' * x' * x' * x' * 1\texttt{>}$$

gen_power **関数によって生成されたコードは** power **関数より高速**

- **関数呼び出しがない**
- **条件式がない**

多段階 let 挿入

- 入れ子になった for ループなどを飛び越えた**コード移動**を許す仕組み
- ループ不変式の移動によって，**効率的なコード生成**に必要なプログラミング技法

for 文を使ったコード生成の例

コード生成器

$$\underline{\textbf{for}}\ x = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$\quad \underline{\textbf{for}}\ y = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$
$$\qquad \underline{\textbf{aryset}}\ a\ (x, y)\ (x + \text{complex calculation})$$

for 文を使ったコード生成の例

コード生成器

$$\underline{\textbf{for}}\ x = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$\quad \underline{\textbf{for}}\ y = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$
$$\qquad \underline{\textbf{aryset}}\ a\ (x, y)\ (x + \text{complex calculation})$$

$$\leadsto^{*}$$

生成されるコード

$$< \textbf{for}\ x = 0\ \textbf{to}\ n\ \textbf{do}$$
$$\quad \textbf{for}\ y = 0\ \textbf{to}\ m\ \textbf{do}$$
$$\qquad a[x, y] \leftarrow (x + \text{complex calculation}) >$$

多段階 let 挿入

生成されるコード

< **for** $x = 0$ **to** n **do**

    for $y = 0$ **to** m **do**

        $a[x, y] \leftarrow x +$ complex calculation>

多段階 let 挿入

生成されるコード

< **for** $x = 0$ **to** n **do**

  let $u =$ complex calculation **in**

  for $y = 0$ **to** m **do**

   $a[x, y] \leftarrow x + u$ >

多段階 let 挿入

生成されるコード

< **for** $x = 0$ **to** n **do**

  let $u =$ complex calculation **in** — u が x にのみ依存し y には依存しない

  for $y = 0$ **to** m **do**

    $a[x, y] \leftarrow x + u$ >

多段階 let 挿入

生成されるコード

< **let** $u =$ complex calculation **in**

for $x = 0$ **to** n **do**

for $y = 0$ **to** m **do**

$a[x, y] \leftarrow x + u$ >

多段階 let 挿入

生成されるコード

< **let** u = complex calculation **in** — u が x にも y にも依存しない式

for $x = 0$ **to** n **do**

for $y = 0$ **to** m **do**

$a[x, y] \leftarrow x + u$ >

そのような多段階の let 挿入を どうやって行うか

多段階 let 挿入

コード生成器

$$\underline{\textbf{for}}\ x = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$\quad \underline{\textbf{for}}\ y = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$
$$\qquad \underline{\textbf{let}}\ u = \text{complex calculation}\ \underline{\textbf{in}}$$
$$\qquad\quad (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (x, y)\ (i + u))$$

多段階 let 挿入

コード生成器

... <u>**for**</u> $x = 0$ <u>**to**</u> n <u>**do**</u>

  ... <u>**for**</u> $y = 0$ <u>**to**</u> m <u>**do**</u>

    ... <u>**let**</u> $u =$ complex calculation <u>**in**</u>

      ... (<u>**aryset**</u> <a> (x, y) $(i + u)$))

多段階 let 挿入

コード生成器

$$\ldots\ \underline{\textbf{for}}\ x = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$\ldots\ \underline{\textbf{for}}\ y = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$
$$\ldots\ \underline{\textbf{let}}\ u = \text{complex calculation}\ \underline{\textbf{in}}$$
$$\ldots\ (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (x, y)\ (i + u))$$

コントロールオペレータ shift0/reset0 の導入

… のところに後に説明する shift0/reset0 というコントロールオペレータを用いることで，多段階 let 挿入を行う

危険な例

危険なコード生成の例

コード生成器

… **for** $x = 0$ **to** n **do**

… **for** $y = 0$ **to** m **do**

… **let** $u =$ complex calculation **in**

… (**aryset** <a> (x, y) $(i + u)$)

危険なコード生成の例

コード生成器

$\ldots$ <u>**for**</u> $x = 0$ <u>**to**</u> n <u>**do**</u>

$\ldots$ <u>**for**</u> $y = 0$ <u>**to**</u> m <u>**do**</u>

$\ldots$ <u>**let**</u> $u =$ complex calculation <u>**in**</u>

$\ldots$ (<u>**aryset**</u> <a> (x, y) $(i + u)$)

$$\rightsquigarrow^*$$

生成されるコード

< **let** $u =$ complex calculation **in** — u が x にも y にも依存する式

for $x = 0$ **to** n **do**

for $y = 0$ **to** m **do**

$a[x, y] \leftarrow (x + u)$ >

危険なコード生成の例

コード生成器

$$\ldots\ \underline{\textbf{for}}\ x = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$

$$\ldots\ \underline{\textbf{for}}\ y = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$

$$\ldots\ \underline{\textbf{let}}\ u = \text{complex calculation}\ \underline{\textbf{in}}$$

$$\ldots\ (\underline{\textbf{aryset}}\ \text{<}a\text{>}\ (x, y)\ (i + u))$$

complex calculation によって挿入できる場所が異なる

- 多段階 let 挿入が可能となっても，安全に挿入できる場所とそうでない場所がある
- 安全に let 挿入を行うためにどうすればよいかを考える必要がある

コード生成の利点と課題

利点

- **「保守性・再利用性の高さ」**と**「実行性能の高さ」**の両立

コード生成の利点と課題

利点

- 「保守性・再利用性の高さ」と「実行性能の高さ」の両立

課題

- 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: 簡潔で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

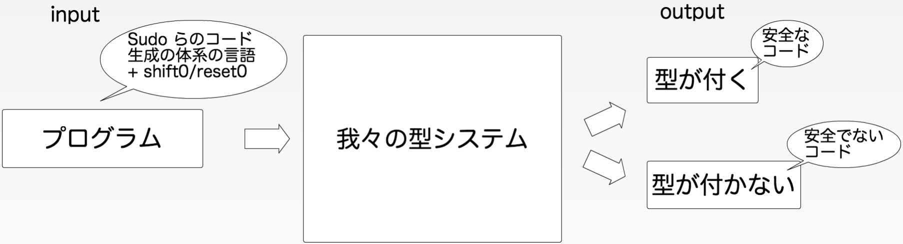
- コントロールオペレータ shift0/reset0 を利用し，let 挿入などのコード生成技法を表現
- 型システムを構築して型安全性を保証

アウトライン

1. 概要
2. 研究の目的
3. **研究の内容**
4. まとめと今後の課題

表現力を上げ（コードレベルでの多段階let挿入），安全性も保証するためにどうすればよいのか

本研究の手法



input
Sudo らのコード
生成の体系の言語
+ shift0/reset0
プログラム
我々の型システム
output
安全な
コード
型が付く
安全でない
コード
型が付かない

まず表現力について

コード生成器と生成されるコード

コード生成器

... $\underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$

... $\underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$

... $\underline{\textbf{let}}\ y = t\ \underline{\textbf{in}}$

... $(\underline{\textbf{aryset}}\ \text{<}a\text{>}\ (i, j)\ b[i] + y)$

$\Rightarrow$

生成されるコード

$\textbf{let}\ y = t\ \textbf{in}$

$\textbf{for}\ i = 0\ \textbf{to}\ n\ \textbf{do}$

$\textbf{for}\ j = 0\ \textbf{to}\ m\ \textbf{do}$

$a[i, j] \leftarrow b[i] + y$

shift0/reset0 による多段階 let 挿入

$$\textbf{reset0}(E[\textbf{shift0}\ k \to e]) \to e\{k \Leftarrow E\}$$

コード生成器

$\ldots\ \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$

$\quad \ldots\ \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$

$\quad\quad \ldots\ \underline{\textbf{let}}\ y = t\ \underline{\textbf{in}}$

$\quad\quad\quad \ldots\ (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (i, j)\ b[i] + y)$

shift0/reset0 による多段階 let 挿入

$$\textbf{reset0}(E[\textbf{shift0}\ k \to e]) \to e\{k \Leftarrow E\}$$

コード生成器

$$\textbf{reset0}\ \ \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$\textbf{reset0}\ \ \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$
$$\textbf{shift0}\ k_2\ \to\ \textbf{shift0}\ k_1\ \to\ \underline{\textbf{let}}\ y = t\ \underline{\textbf{in}}$$
$$k_1\ (k_2\ (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (i, j)\ b[i] + y))$$

shift0/reset0 による多段階 let 挿入

$$\textbf{reset0}(E[\textbf{shift0}\ k \to e]) \to e\{k \Leftarrow E\}$$

コード生成器

$$\begin{array}{l}
\textbf{reset0}\ \ \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}} \\
\quad \textbf{reset0}\ \ \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}} \\
\quad\quad \textbf{shift0}\ k_2\ \to\ \textbf{shift0}\ k_1\ \to\ \underline{\textbf{let}}\ y = t\ \underline{\textbf{in}} \\
\quad\quad\quad k_1\ (k_2\ (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (i, j)\ b[i] + y))
\end{array}$$

$$\begin{aligned}
k_1 &= \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}} \\
k_2 &= \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}
\end{aligned}$$

shift0/reset0 による多段階 let 挿入

$$\textbf{reset0}(E[\textbf{shift0}\ k \to e]) \to e\{k \Leftarrow E\}$$

コード生成器

$$\underline{\textbf{let}}\ y = t\ \underline{\textbf{in}}$$
$$\quad k_1\ (k_2\ (\underline{\textbf{aryset}}\ \texttt{<}a\texttt{>}\ (i, j)\ b[i] + y))$$

$$k_1 = \quad \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}}$$
$$k_2 = \quad \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}}$$

shift0/reset0 による多段階 let 挿入

$$\textbf{reset0}(E[\textbf{shift0}\ k \to e]) \to e\{k \Leftarrow E\}$$

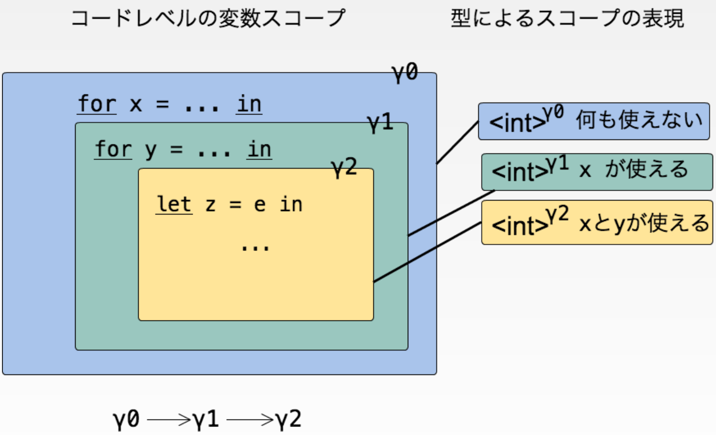
生成されるコード

$$
\begin{array}{l}
\textbf{let}\ y\ =\ t\ \textbf{in} \\
\quad \textbf{for}\ i = 0\ \textbf{to}\ n\ \textbf{do} \\
\quad\quad \textbf{for}\ j = 0\ \textbf{to}\ m\ \textbf{do} \\
\quad\quad\quad a[i,j] \leftarrow b[i] + y
\end{array}
$$

次に安全性

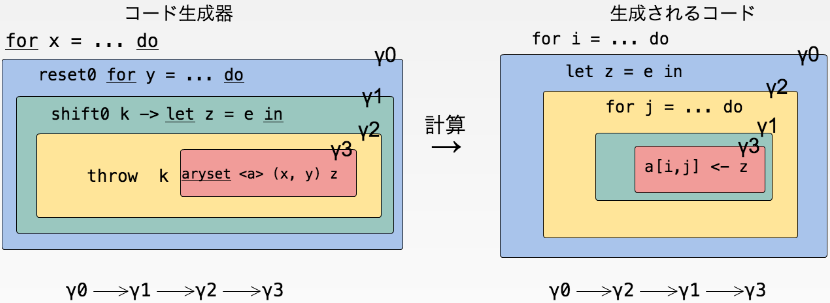
コード生成前の段階で，安全なコードかどうかを判断する

環境識別子 EC によるスコープ表現 [Taha+2003] [Sudo+2014]

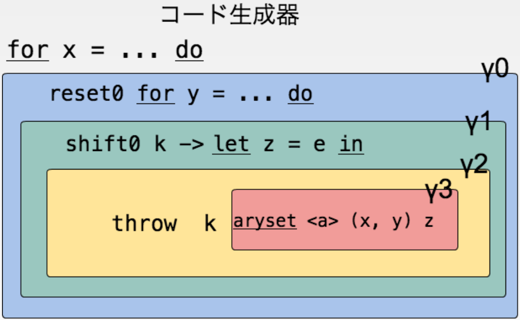


γ_i…Refined Environment Classifier

EC の洗練化　（本研究）


コード生成器
for x = ... do
γ0
reset0 for y = ... do
γ1
shift0 k -> let z = e in
γ2
γ3
throw k aryset <a> (x, y) z
γ0 ⟶γ1 ⟶γ2 ⟶γ3
計算
⟶
生成されるコード
for i = ... do
γ0
let z = e in
γ2
for j = ... do
γ1
γ3
a[i,j] <- z
γ0 ⟶γ2 ⟶γ1 ⟶γ3

ECの洗練化 （本研究）

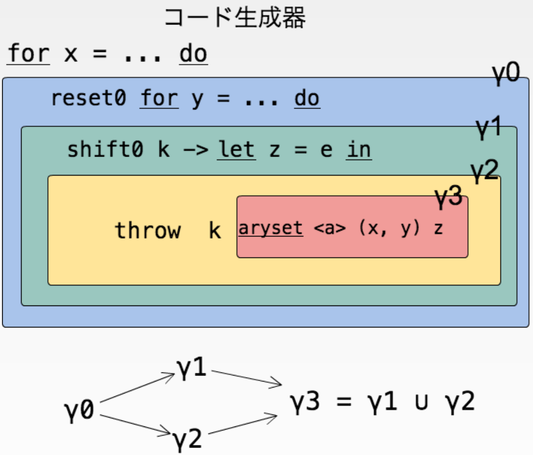


$\gamma 0 \longrightarrow \gamma 1 \longrightarrow \gamma 2 \longrightarrow \gamma 3$

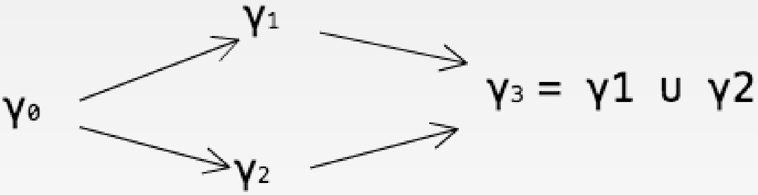
かつ

$\gamma 0 \longrightarrow \gamma 2 \longrightarrow \gamma 1 \longrightarrow \gamma 3$

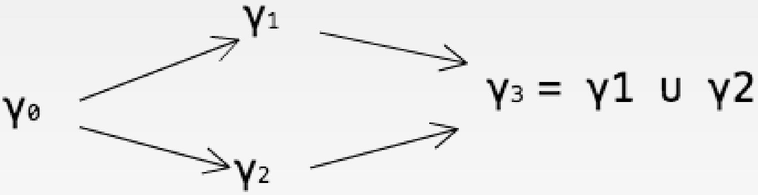
ECの洗練化 （本研究）


コード生成器
for x = ... do
γ0
reset0 for y = ... do
γ1
shift0 k -> let z = e in
γ2
γ3
throw k aryset <a> (x, y) z
γ0
γ1
γ2
γ3 = γ1 ∪ γ2

EC のジョイン

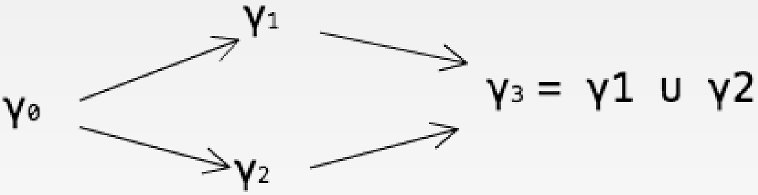


EC のジョイン



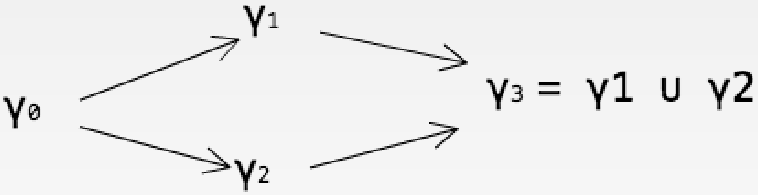
- γ_1 のコードレベル変数は γ_2 では使えない

EC のジョイン



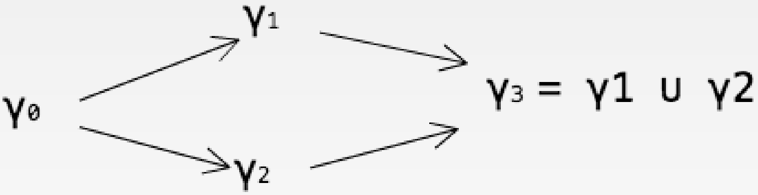
- γ_1 のコードレベル変数は γ_2 では使えない
- γ_2 のコードレベル変数は γ_1 では使えない

EC のジョイン



- γ_1 のコードレベル変数は γ_2 では使えない
- γ_2 のコードレベル変数は γ_1 では使えない
- γ_1, γ_2 のコードレベル変数は γ_3 で使える

EC のジョイン



- γ_1 のコードレベル変数は γ_2 では使えない
- γ_2 のコードレベル変数は γ_1 では使えない
- γ_1, γ_2 のコードレベル変数は γ_3 で使える

$\Rightarrow$ Sudo らの体系に $\cup$ を追加

コード生成+shift0/reset0 の型システム (の一部)

コードレベルのラムダ抽象:

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

reset0:

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

shift0:

$$\frac{\Gamma,\ k : (\langle t_1 \rangle^{\gamma_1} \Rightarrow \langle t_0 \rangle^{\gamma_0})\sigma \vdash e : \langle t_0 \rangle^{\gamma_0} ; \sigma \quad \Gamma \models \gamma_1 \geq \gamma_0}{\Gamma \vdash \textbf{shift0}\ k.e : \langle t_1 \rangle^{\gamma_1}\ ;\ \langle t_0 \rangle^{\gamma_0}, \sigma}$$

throw:

$$\frac{\Gamma \vdash v : \langle t_1 \rangle^{\gamma_1 \cup \gamma_2} ; \sigma \quad \Gamma \models \gamma_2 \geq \gamma_0}{\Gamma,\ k : (\langle t_1 \rangle^{\gamma_1} \Rightarrow \langle t_0 \rangle^{\gamma_0})\sigma \vdash \textbf{throw}\ k\ v : \langle t_0 \rangle^{\gamma_2} ; \sigma}$$

型が付く例/付かない例

コード生成器

$$
\begin{array}{l}
e = \textbf{reset0}\ \ \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}} \\
\quad \textbf{reset0}\ \ \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}} \\
\quad\quad \textbf{shift0}\ k_2\ \rightarrow\ \textbf{shift0}\ k_1\ \rightarrow\ \underline{\textbf{let}}\ y = t\ \underline{\textbf{in}} \\
\quad\quad\quad k_1\ (k_2\ (\underline{\textbf{aryset}}\ \text{<}a\text{>}\ (i, j)\ b[i] + y))
\end{array}
$$

型が付く例/付かない例

コード生成器

$$
\begin{aligned}
e = {} & \textcolor{red}{\textbf{reset0}}\ \ \underline{\textbf{for}}\ i = 0\ \underline{\textbf{to}}\ n\ \underline{\textbf{do}} \\
& \quad \textcolor{blue}{\textbf{reset0}}\ \ \underline{\textbf{for}}\ j = 0\ \underline{\textbf{to}}\ m\ \underline{\textbf{do}} \\
& \quad\ \ \textcolor{blue}{\textbf{shift0}\ k_2} \ \to\ \textcolor{red}{\textbf{shift0}\ k_1} \ \to\ \underline{\textbf{let}}\ y = t\ \underline{\textbf{in}} \\
& \quad\quad \textcolor{red}{k_1}\ (\textcolor{blue}{k_2}\ (\underline{\textbf{aryset}}\ <a>\ (i,j)\ b[i] + y))
\end{aligned}
$$

生成されるコード

×

$$
\begin{aligned}
e \leadsto^* {} & \textbf{let}\ y\ =\ a[i][j]\ \textbf{in} \\
& \quad \textbf{for}\ i = 0\ \textbf{to}\ n\ \textbf{do} \\
& \quad\ \ \textbf{for}\ j = 0\ \textbf{to}\ m\ \textbf{do} \\
& \quad\quad a[i,j] \leftarrow b[i] + y
\end{aligned}
$$

○

$$
\begin{aligned}
e \leadsto^* {} & \textbf{let}\ y\ =\ 7\ \textbf{in} \\
& \quad \textbf{for}\ i = 0\ \textbf{to}\ n\ \textbf{do} \\
& \quad\ \ \textbf{for}\ j = 0\ \textbf{to}\ m\ \textbf{do} \\
& \quad\quad a[i,j] \leftarrow b[i] + y
\end{aligned}
$$

あれもこれもできる

アウトライン

まとめと今後の課題

まとめ

- コード生成言語の型システムに shift0/reset0 を組み込んだ型システムの設計を完成させた.
- 安全なコードの場合に型が付くこと，安全でないコードの場合には型が付かないように意図通りに型システムが設計できていることをみた

今後の課題

- 設計した型システムの健全性の証明（Subject recudtion 等）
- 型推論アルゴリズムの開発
- 言語の拡張
  - グローバルな参照 (OCaml の let ref)
  - 生成したコードの実行 (MetaOCaml の run)

APPENDIX

アウトライン

5. 健全性の証明

健全性の証明 (Subject Reduction)

型安全性 (型システムの健全性; Subject Reduction **等の性質) を厳密に証明する.**

Subject Redcution Property

$\Gamma \vdash M : \tau$ **が導ければ (プログラム** M **が型検査を通れば),** M **を計算して得られる任意の** N **に対して,** $\Gamma \vdash N : \tau$ **が導ける (**N **も型検査を通り,** M **と同じ型, 同じ自由変数を持つ)**