オンライン／オフライン
部分評価の混成的な方式

小林研究室
修士２年
住井 英二郎

発表の概要

- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

発表の概要

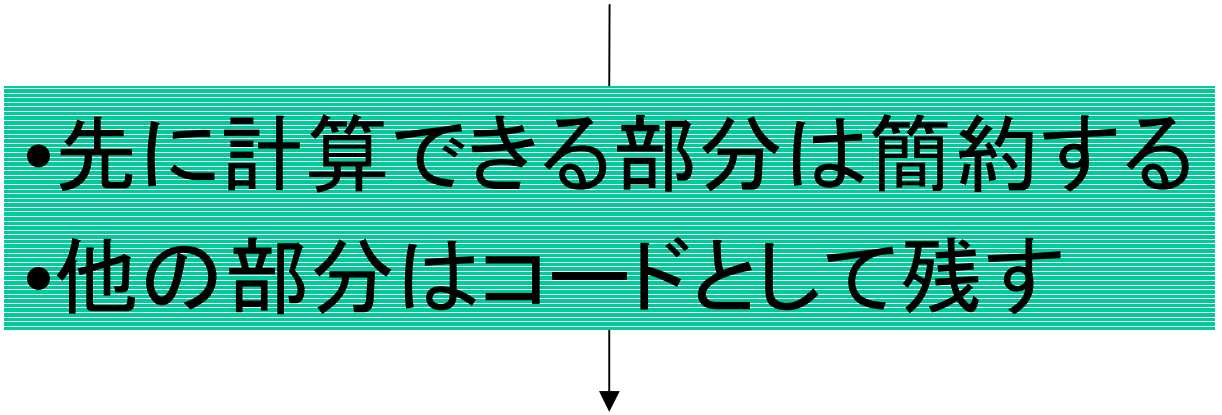
- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

部分評価とは

プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$

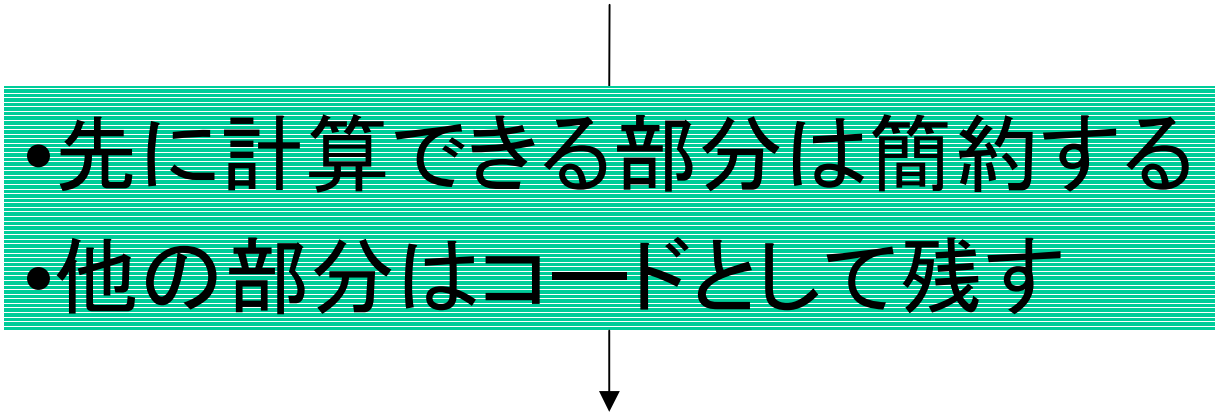
部分評価とは

プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$



部分評価とは

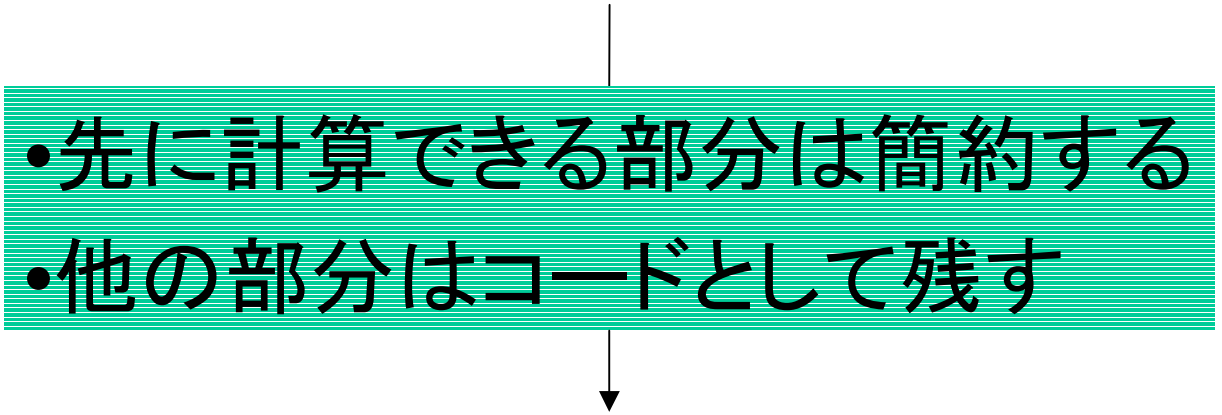
プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$



特化されたプログラム $p_{x0}(y)$

部分評価とは

プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$

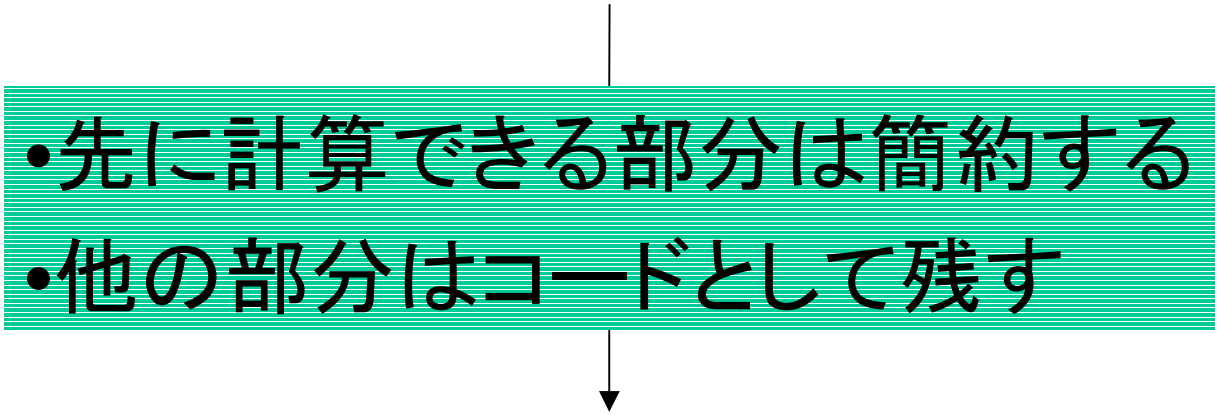


特化されたプログラム $p_{x0}(y)$

例： $p(x, y) = 1 + x + y, \quad x = 2$

部分評価とは

プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$



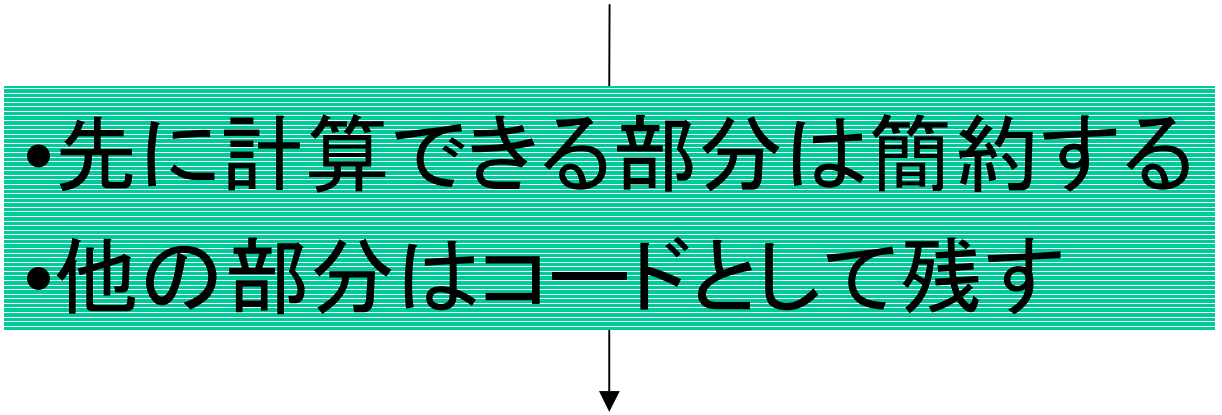
特化されたプログラム $p_{x0}(y)$

例： $p(x, y) = 1 + x + y, \quad x = 2$

$\rightarrow \quad p_2(y) = 1 + 2 + y$

部分評価とは

プログラム $p(x, y)$ ＋ 入力の一部 $x = x_0$



特化されたプログラム $p_{x0}(y)$

例： $p(x, y) = 1 + x + y, \quad x = 2$

$\rightarrow \quad p_2(y) = 3 + y$

我々の目標

1. 特化されたプログラムの効率
   （～ コンパイルされたプログラムの効率）
2. 特化自体の効率
   （～ コンパイル自体の効率）

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）

([Ruf 93] 等)

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）

$p_2(y) = 1 + 2 + y$

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）

$$p_2(y) = (\textbf{if } (\mathit{known?}\ 1)\ \textbf{and}\ (\mathit{known?}\ 2)\ \textbf{then}\ 1\ \overline{+}\ 2\ \textbf{else}\ 1\ \underline{+}\ 2) + y$$

$\overline{+}$：簡約する　$\underline{+}$：コードとして残す

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）

$$p_2(y) = 3 + y$$

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める（オンライン部分評価）

$p_2(y) = \textbf{if}\ (\mathit{known?}\ 3)\ \textbf{and}\ (\mathit{known?}\ y)$
$\textbf{then}\ 3\ \overline{+}\ y\ \textbf{else}\ 3\ \underline{+}\ y$

$\overline{+}$：簡約する　$\underline{+}$：コードとして残す

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）

$p_2(y) = 3 \pm y$

$\pm$：簡約する　$\pm$：コードとして残す

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）
- 静的解析を行って、特化の前に決める
  （オフライン部分評価）

([Bondorf 90] [Gomard 92] 等)

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）
- 静的解析を行って、特化の前に決める
  （オフライン部分評価）

$$p_2(y) = 1 + 2 + y$$

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）
- 静的解析を行って、特化の前に決める
  （オフライン部分評価）

$$p_2(y) = 1 \overline{+} 2 \underline{+} y$$

$\overline{+}$：簡約する　$\underline{+}$：コードとして残す

従来の方式

「簡約するか、コードとして残すか」を

- 値を見ながら、特化の最中に決める
  （オンライン部分評価）
- 静的解析を行って、特化の前に決める
  （オフライン部分評価）

$p_2(y) = 3 \pm y$

$\pm$：簡約する　$\pm$：コードとして残す

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率

$$\textbf{let}\ f(z) = z + 1\ \ \textbf{in}\ \langle f(2), f(y)\rangle$$

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率

$$\textbf{let}\ f(z) = z \underline{\pm} 1\ \ \textbf{in}\ \langle f(2), f(y)\rangle$$

$\overline{+}$：簡約する　$\underline{+}$：コードとして残す

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率

$$\textbf{let}\ f(z) = z \underline{\pm} 1\ \ \textbf{in}\ \langle f(2), f(y)\rangle$$
$$\rightarrow\ \langle 2 \underline{\pm} 1, y \underline{\pm} 1\rangle$$

$\underline{-}$: 簡約する　$\underline{+}$: コードとして残す

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率
- 両者: コードの複製を防ぐための
  "let-insertion" のオーバーヘッドが大きい

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率
- 両者: コードの複製を防ぐための
  "let-insertion" のオーバーヘッドが大きい

$$(\lambda v.\, \langle v, v, v \rangle)\ \ (1 + 2 + y)$$

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率
- 両者： コードの複製を防ぐための
  "let-insertion" のオーバーヘッドが大きい

$$(\lambda v.\, \langle v, v, v\rangle)\ \ (1 \overline{+} 2 \pm y)$$
$$\rightarrow\ \ \langle 3 \pm y, 3 \pm y, 3 \pm y\rangle$$

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率
- 両者： コードの複製を防ぐための
  "let-insertion" のオーバーヘッドが大きい

$$(\lambda v.\, \langle v, v, v\rangle)\ \ (\text{insert-let}(1 \overline{+} 2 \underline{+} y))$$
$$\rightarrow\ \ \textbf{let}\ w = 3 \underline{+} y\ \textbf{in}\ \langle w, w, w\rangle$$

問題点

- オンライン部分評価（～ dynamic typing）:
  静的な情報を用いない
  ⇒ 特化自体が非効率
- オフライン部分評価（～ static typing）:
  静的な情報しか用いない
  ⇒ 特化されたプログラムが非効率
- 両者: コードの複製を防ぐための
  "let-insertion" のオーバーヘッドが大きい

（コードとして残すすべての式を
let で変数に束縛する）

問題点

	従来の オンライン 部分評価	従来の オフライン 部分評価
特化されたプログラムの効率	○ $\langle 3, y \pm 1 \rangle$	△ $\langle 1 \pm 2, y \pm 1 \rangle$
特化自体の効率	× **if** (*known? i*) **and** (*known? j*) **then** $i \overline{+} j$ **else** insert-let($i \pm j$)	△ $i \overline{+} j$ または insert-let($i \pm j$)

発表の概要

- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

我々の方式

オンライン部分評価のオーバーヘッドを削減

- 部分的な静的解析（～ soft typing）
- 継続のかわりに状態を用いた let-insertion
- Cogen アプローチ

我々の方式

オンライン部分評価のオーバーヘッドを削減

- 部分的な静的解析（～ soft typing）
- 継続のかわりに状態を用いた let-insertion
- Cogen アプローチ

部分的な静的解析

– "known"かどうかを解析

⇒「*known?*」を削減

部分的な静的解析

- "known"かどうかを解析
  ⇒「*known?*」を削減
- コードとして残る回数を解析
  ⇒ let-insertion を削減

部分的な静的解析

- "known"かどうか

  （0: never, ω: always, T: どちらともいえない）
- コードとして残る回数

  （0: 0回, 1: 1回以下, ω: 任意回）

型の一部として推論

例.　$1 : \mathrm{int}^{(\omega,0)} + 2 : \mathrm{int}^{(\omega,0)}$

特化の流れ

特化の流れ

元のプログラム

特化の流れ

元のプログラム

解析

特化の流れ

元のプログラム

解析

注記のついたプログラム

特化の流れ

元のプログラム

解析

注記のついたプログラム

特化器生成器

特化の流れ

元のプログラム

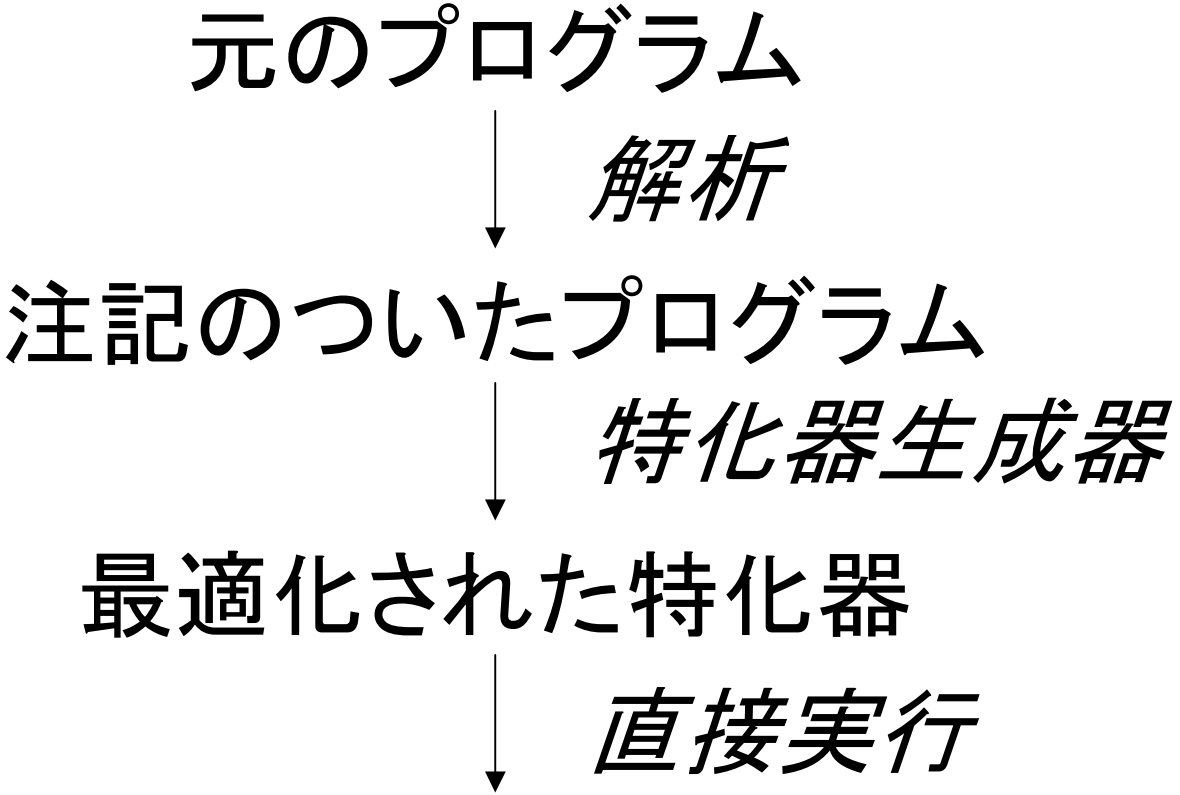
解析

注記のついたプログラム

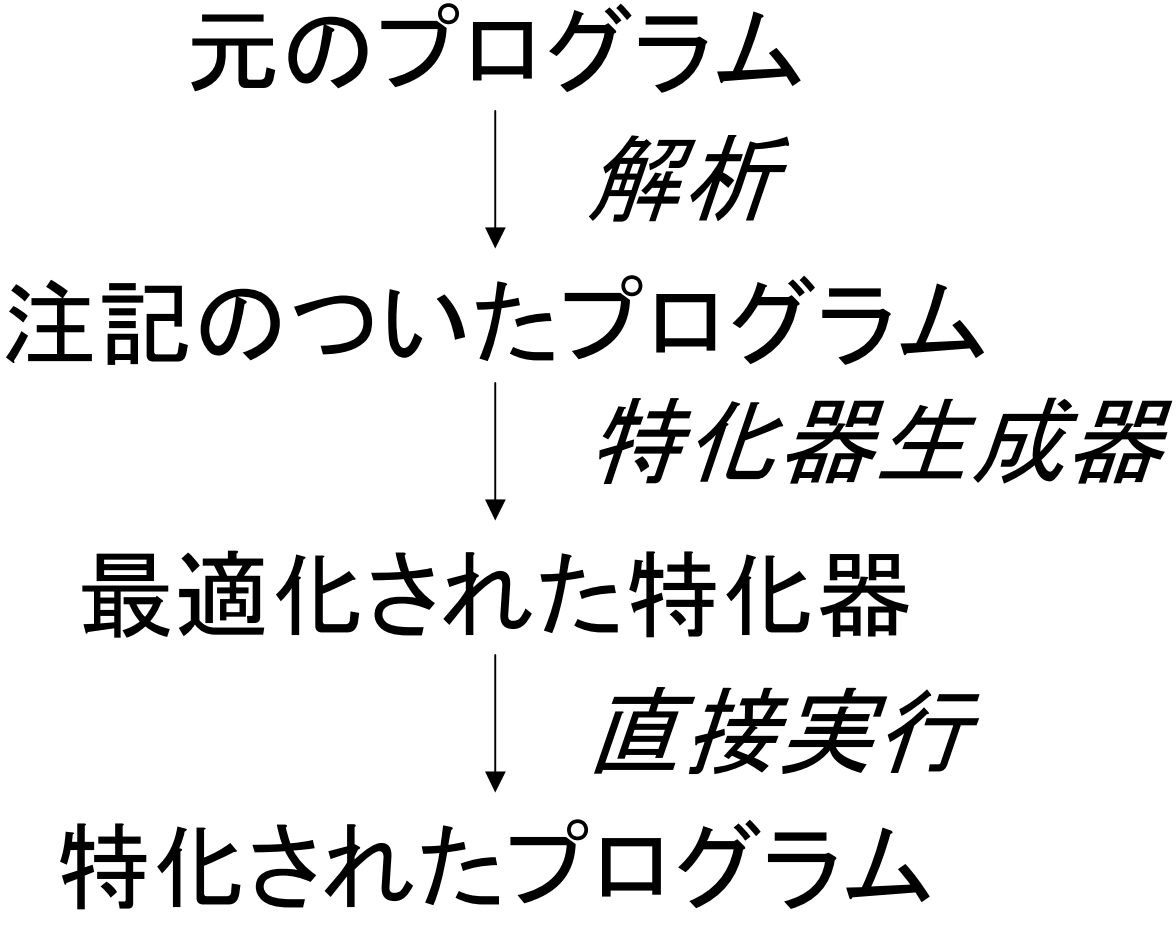
特化器生成器

最適化された特化器

特化の流れ



特化の流れ



特化の例（1）

y : unknown

- $1 + 2 + y$

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} + 2 : \mathrm{int}^{(\omega,0)} + y : \mathrm{int}^{(0,1)}$
  $\Rightarrow 1 \overline{+} 2 \underline{\pm} y$

$\overline{+}$: 簡約する　$\underline{\pm}$: コードとして残す

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} + 2 : \mathrm{int}^{(\omega,0)} + y : \mathrm{int}^{(0,1)}$

  $\Rightarrow 1 \overline{+} 2 \pm y$

  $\rightarrow 3 \pm y$

$\overline{+}$: 簡約する　$\pm$: コードとして残す

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$
  $\Rightarrow \ 1 \overline{+} 2 \pm y$
  $\rightarrow \ 3 \pm y$
- **let** $f(z) = z + 1$ **in** $\langle f(2), f(y) \rangle$

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$
  $\Rightarrow \ 1 \overline{+} 2 \pm y$
  $\rightarrow \ 3 \pm y$
- **let** $f\,(z : \mathrm{int}^{(\mathrm{T},1)}) = z + 1$ **in** $\langle f\,(2), f\,(y)\rangle$

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$
  $\Rightarrow \ 1 \,\overline{+}\, 2 \pm y$
  $\rightarrow \ 3 \pm y$
- **let** $f\,(z : \mathrm{int}^{(\mathrm{T},1)}) = z + 1$ **in** $\langle f\,(2), f\,(y)\rangle$
  $\Rightarrow$ **let** $f\,(z) =$ **if** $(\mathit{known?}\ z)$ **then** $z \,\overline{+}\, 1$ **else** $z \,\underline{\pm}\, 1$
  in $\langle f\,(2), f\,(y)\rangle$

$\overline{+}$: 簡約する　$\underline{\pm}$: コードとして残す

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$
  $\Rightarrow \ 1 \mathbin{\overline{+}} 2 \pm y$
  $\rightarrow \ 3 \pm y$
- **let** $f\,(z : \mathrm{int}^{(\mathrm{T},1)}) = z + 1$ **in** $\langle f\,(2), f\,(y)\rangle$
  $\Rightarrow$ **let** $f\,(z) =$ **if** $(\mathit{known?}\ z)$ **then** $z \mathbin{\overline{+}} 1$ **else** $z \mathbin{\underline{\pm}} 1$
  in $\langle f\,(2), f\,(y)\rangle$
  $\rightarrow \langle 2 \mathbin{\overline{+}} 1, y \mathbin{\underline{\pm}} 1\rangle$

$\overline{+}$: 簡約する　$\underline{\pm}$: コードとして残す

特化の例（1）

y : unknown

- $1 : \mathrm{int}^{(\omega,0)} \ + \ 2 : \mathrm{int}^{(\omega,0)} \ + \ y : \mathrm{int}^{(0,1)}$
  $\Rightarrow \ 1 \mathbin{\overline{+}} 2 \pm y$
  $\rightarrow \ 3 \pm y$
- **let** $f\,(z : \mathrm{int}^{(\mathrm{T},1)}) = z + 1$ **in** $\langle f\,(2), f\,(y)\rangle$
  $\Rightarrow$ **let** $f\,(z) =$ **if** $(\mathit{known?}\ z)$ **then** $z \mathbin{\overline{+}} 1$ **else** $z \mathbin{\underline{\pm}} 1$
  in $\langle f\,(2), f\,(y)\rangle$
  $\rightarrow \langle 3, y \mathbin{\underline{\pm}} 1\rangle$

$\overline{+}$: 簡約する　$\underline{\pm}$: コードとして残す

特化の例（２）

- $(\lambda v.\ v)\ \ (1 + 2 + y)$

特化の例（２）

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$

特化の例(2)

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \bar{+} 2 \underline{+} y)$

$\bar{+}$: 簡約する　$\underline{+}$: コードとして残す

特化の例(2)

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \overline{+} 2 \underline{+} y)$
  $\rightarrow\ 3 \underline{+} y$

$\overline{+}$: 簡約する　$\underline{+}$: コードとして残す

特化の例（2）

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \overline{+} 2 \pm y)$
  $\rightarrow\ 3 \pm y$
- $(\lambda v.\ \langle v, v, v \rangle)\ \ (1 + 2 + y)$

特化の例（2）

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \mp 2 \pm y)$
  $\rightarrow\ 3 \pm y$
- $(\lambda v : \mathrm{int}^{(0,\omega)}.\ \langle v, v, v \rangle)\ \ (1 + 2 + y)$

特化の例(2)

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \mathbin{\overline{+}} 2 \mathbin{\underline{\pm}} y)$
  $\rightarrow\ 3 \mathbin{\underline{\pm}} y$
- $(\lambda v : \mathrm{int}^{(0,\omega)}.\ \langle v, v, v\rangle)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ \langle v, v, v\rangle)\ (\textit{insert-let}(1 \mathbin{\overline{+}} 2 \mathbin{\underline{\pm}} y))$

$\overline{+}$: 簡約する　$\underline{\pm}$: コードとして残す

特化の例（2）

- $(\lambda v : \mathrm{int}^{(0,1)}.\ v)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ v)\ (1 \mathbin{\overline{+}} 2 \mathbin{\underline{+}} y)$
  $\rightarrow\ 3 \mathbin{\underline{+}} y$
- $(\lambda v : \mathrm{int}^{(0,\omega)}.\ \langle v, v, v\rangle)\ \ (1 + 2 + y)$
  $\Rightarrow\ (\lambda v.\ \langle v, v, v\rangle)\ (\textit{insert-let}(1 \mathbin{\overline{+}} 2 \mathbin{\underline{+}} y))$
  $\rightarrow\ \mathbf{let}\ w = 3 \mathbin{\underline{+}} y\ \mathbf{in}\ \langle w, w, w\rangle$

$\overline{+}$：簡約する　$\underline{+}$：コードとして残す

発表の概要

- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

比較した方式

- 単純なオンライン部分評価： (T,ω) のみ
  （＋ 後処理によるインライン化）
- 単純なオフライン部分評価： $(\omega,0)$ と $(0,\omega)$
  （＋ 手動による束縛時改善）
- [Sperber-96]： $(\omega,0)$, $(0,\omega)$, および (T,ω)
- 我々の方式

使用したアプリケーション

- imp

  単純な手続き型言語のインタプリタ
- match1

  パターンマッチング（パターンについて特化）
- match2

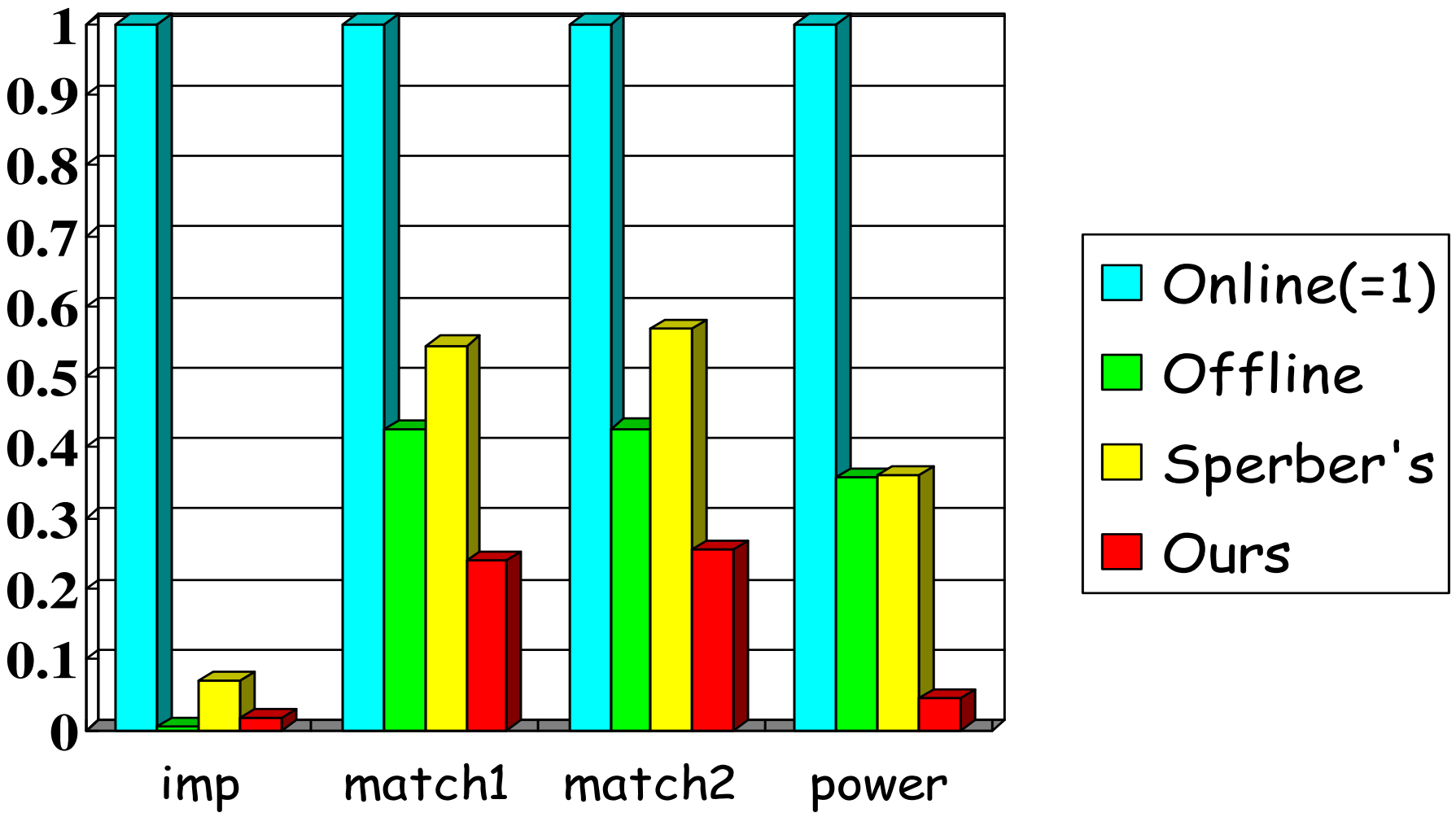
  パターンマッチング（文字列について特化）
- power

  べき乗

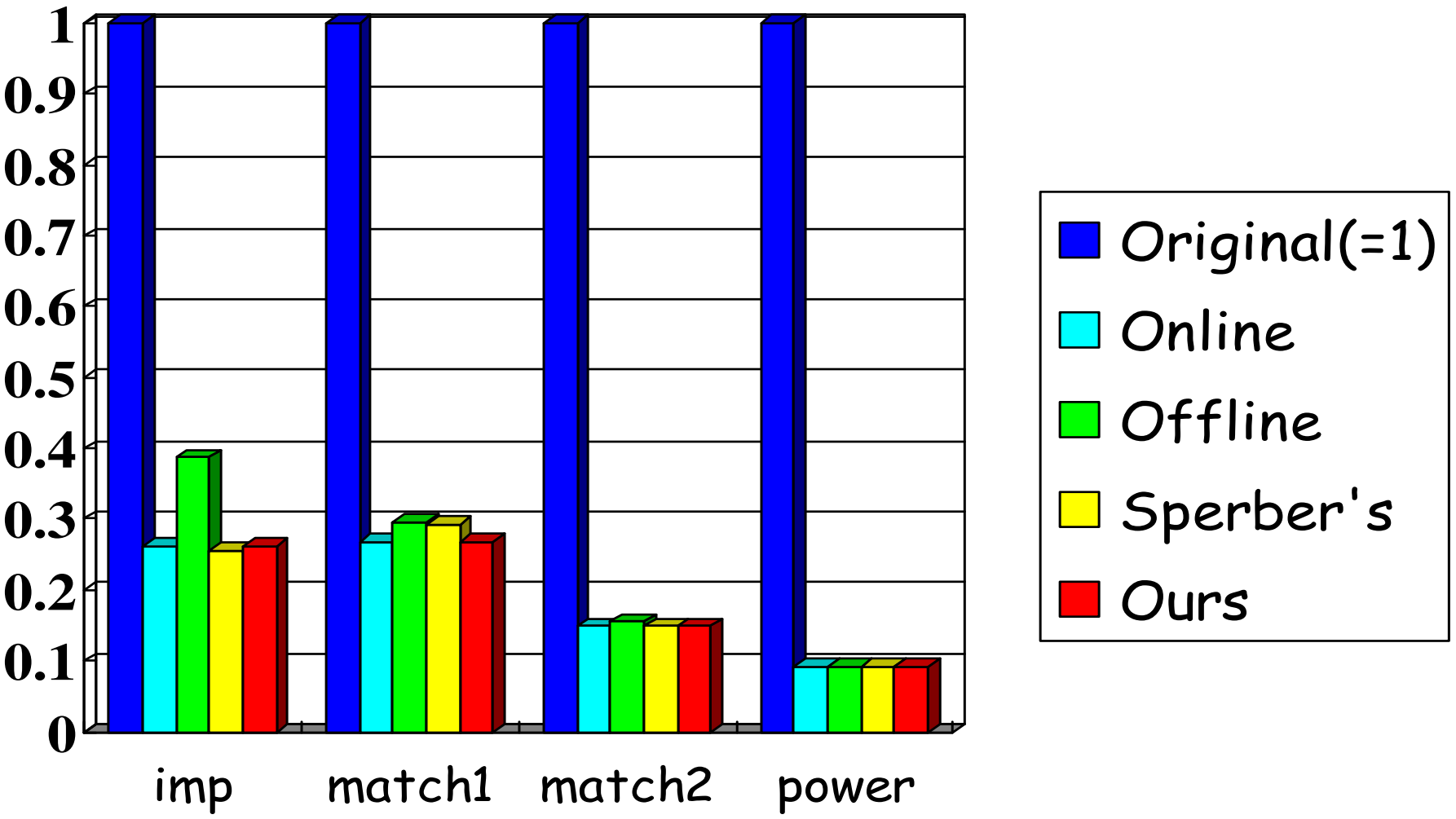
実行した環境

- Mobile Pentium II 400MHz
- 128MB Main Memory
- Linux 2.2.10
- SML/NJ 110.0.3

特化自体の効率


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
imp
match1
match2
power
Online(=1)
Offline
Sperber's
Ours

特化されたプログラムの効率



発表の概要

- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

既存の解析

- Binding-Time Analysis ([Henglein 91] 等)

  我々の **(ω,0)** と **(0,ω)** に相当
- BTA with "Unknown" [Sperber 96]

  我々の **(T,ω)** に相当
- BTA with "Both" [浅井 99]

  我々の **(ω,ω)** に相当
- Abstract Occurrence Counting Analysis [Bondorf 90]

  我々の **(0,1)** にほぼ相当

我々はこれらを単一の解析で包摂

発表の概要

- 背景と問題点
- 我々の方式
- 実験結果
- 関連研究
- 結論

要約

	従来の オンライン 部分評価	従来の オフライン 部分評価	我々の方式
特化された プログラムの 効率	○	△	○
特化自体の 効率	×	△	○

今後の展望

- Cogen Approach の一般化
  インタプリタをメタ言語に埋め込める
- State-Based Let-Insertion の一般化？
  継続操作を状態操作で置き換えられる
  when & how?

修士二年間の研究業績

- 査読付き会議論文
  - ***A Generalized Deadlock-Free Process Calculus*** **(HLCL'98)**
  - ***Online Type-Directed Partial Evaluation for Dynamically-Typed Languages*** **(PPL'99)**
  - ***Online-and-Offline Partial Evaluation: A Mixed Approach*** **(PEPM'00)**
- ジャーナル論文
  - 動的に型付けされた言語のための型誘導部分評価 （コンピュータソフトウェア）
- その他　口頭発表等　多数

Frequently Asked Questions (1)

[Q] 特化自体の効率は問題にならないのでは？

[A] （特にオンライン部分評価では）問題になる。
オフライン部分評価や自己適用は
なんのために研究されてきたのか？

Frequently Asked Questions (2)

[Q] 解析のコストは？

[A] 問題にならない。

- 解析は1回、特化はn回
- 型推論における制約変数の数と
  同数以下の反復で停止
- 途中で中止できるので、
  「解析のコスト」と「特化自体の効率」を
  自由に調節できる
  (「特化されたプログラムの効率」には
  影響しない)

Frequently Asked Questions (3)

[Q] もっと大きなアプリケーションで実験すべきでは？

[A] Maybe.

ただし、大きなアプリケーションのほうが従来の方式に不利なことに注意。

Frequently Asked Questions (4)

[Q] コードは複製したほうが特化できる場合もあるのでは？

[A] No.

- Let-insertionは「コードとして残す」際の話で、「簡約する」際には無関係
- 特化の後で let を inline するのは、本研究とは別問題（cf. [Ashley 97] 等）

Frequently Asked Questions (5)

[Q] 副作用は扱えるか？

[A] Yes.

- ただし、すべての副作用を
  （簡約しないで）コードとして残している。
- 副作用の簡約は future work だが、
  effect system を取り入れればできるかも？
  (cf. [Thiemann & Dussart])

Frequently Asked Questions (6)

[Q] オフラインのほうが Reasoning がしやすいのでは？

[A] Maybe.

ただし、我々の解析はオフライン部分評価も包摂していることに注意。

Frequently Asked Questions (7)

[Q] "*known?*" の削減と let-insertion の削減は別問題では？

[A] 原理的にはそうだが、実は

一緒に解析するほうが
別々に解析するよりも簡単

である。

Frequently Asked Questions (8)

[Q] 特化自体の停止性は？

[A] 保証されていない。従来の方式でも困難。

Frequently Asked Questions (9)

[Q]「コードとして残る回数」の解析は trivial なのでは？

[A] No. 高階関数があるので。

Frequently Asked Questions (10)

[Q] 既存の解析の単なる寄せ集めなのでは？

[A] No.

最初から単一の解析として、
型システムにより明確に定式化されている。