Online Type-Directed Partial Evaluation for Dynamically-Typed Languages

Eijiro Sumii and Naoki Kobayashi

University of Tokyo

Partial Evaluation

p: プログラム, **s**: 入力の一部

$$\textit{peval}(p, s) = p_s$$

d: 残りの入力

$$\textit{eval}(p, (s, d)) = \textit{eval}(p_s, d)$$

Type-Directed PE

- 古典的な PE (Syntax-Directed PE)

p と **s** の式 (expression)

↓ ***peval***

$\mathbf{p_s}$ の式

- Type-Directed PE [Danvy 96]

p と **s** の式

↓ ***eval*** + type inference

λ**d. p(s, d)** の値 (value) と型

↓ ***reify***

$\mathbf{p_s}$ の式

Type-Directed PE

- 古典的な PE (Syntax-Directed PE)

素朴な方法だと interpretive overhead が大きく遅い

p と **s** の式 (expression)
↓ ***peval***
$\mathbf{p_s}$ の式

- Type-Directed PE [Danvy 96]

どちらも直接の実行なので速い

p と **s** の式
↓ ***eval*** + type inference
λ**d. p(s, d)** の値 (value) と型
↓ ***reify***
$\mathbf{p_s}$ の式

従来の TDPE の問題点

- 適用できる言語 $(\circledR, \acute{}, +)$が限定
  - × 再帰型や動的型を持つ言語
- 与えられた型に盲従
  - ⇨ PE の処理と結果の両方が冗長
- SDPE との関係が不明確
  - ⇨ 従来の技術の応用が困難

我々の貢献

- 様々な関数型言語に適用でき、
  冗長性のない TDPE を提案
- Higher-Order Abstract Syntax を用いた
  ある単純な SDPE から、deforestation で
  我々の TDPE を導出

  ⇒ より複雑な SDPE からは
  より高度な TDPE が得られる

発表の概要

- Introduction
- Danvy の TDPE
- 我々の TDPE
- 比較実験
- SDPE との関係
- 関連研究
- 結論

Danvy の TDPE: Reification

reify = 「eval の逆」

値 **v** とその型 t ↦ ***eval*(e) = v** なる式 **e** の強正規形

⇑

関数の場合が問題（他の場合は自明）

reify **(**r ® s**) v** :=

(1) 引数の「代表」となる r 型の値を生成

(2) 生成した値に **v** を適用

(3) 適用した結果である s 型の値を reify

Danvy の TDPE: Reification の例 (1)

例：***reify*** **(**a ® b ® a**) v**

関数適用

where **v** = l**a.** l**b. ((**l**c. c) @ a)**

v @ x = l**b. ((**l**c. c) @ x)**

シンボル

v @ x @ y = **(**l**c. c) @ x**

= **x**

λ 抽象の syntax constructor

よって

reify **(**a ® b ® a**) v** = l**x.** l**y. x**

Danvy の TDPE: Reification の例 (2)

例：***reify*** **((int ® a) ® a) v**

where **v** = **l a. (a @ (1 + 2))**

v @ x = x @ (1 + 2) ⇒ Error!

そこで **x** を **l b. x @ b** とη 展開して

関数適用の syntax constructor

v @ (l b. x @ b) = (l b. x @ b) @ (1 + 2)
= x @ 3

よって

reify **((int ® a) ® a) v = l x. x @ 3**

Danvy の TDPE: Reification の一般的な定義

reify **:** ***type*** **®** ***value*** **®** ***exp***

reify a **v = v**

reify **(**a **®** b**) v =** l**x. (*reify*** b **(v @ (*reflect*** a **x)))**

(where **x** *is fresh)*

reflect **:** ***type*** **®** ***exp*** **®** ***value***

reflect a **e = e**

reflect **(**a **®** b**) e =** l **a. (*reflect*** b **(e @ (*reify*** a **a)))**

Danvy の TDPE: 問題点

- 適用できる言語が限定されている

  整数型や再帰型には「代表」となる値がない

  – ***reflect int x = ???***

  – ***reflect (a list) y = ???***

- 特化時も実行時も効率が悪い

  型が冗長だと、無駄な reflection が発生して
  結果も冗長になる

  reify ((a ® a) ® a ® a) (l a. a) = l x. l y. (x @ y)

我々の TDPE：Basic Idea (1)

Reflection を行わない！

$$\mathit{reify}(\lambda x.\ t) = \lambda y.\ \mathit{reify}((\lambda x.\ t)\ @\ y) \quad \text{(where } y \text{ is fresh)}$$

$$\mathit{reify}(t_1\ @\ t_2) = \mathit{reify}(t_1)\ @\ \mathit{reify}(t_2)$$

$$\mathit{reify}(y) = y$$

我々の TDPE：Basic Idea (2)

primitive な value destructor たちを、

「式に対しては式を返す」

ように拡張する。

例：関数適用

$\mathbf{e_1 @¢ e_2 = e_1 @ e_2}$

（e_1 が関数の場合）

$\mathbf{e_1 @¢ e_2 = e_1 @ (reify\ e_2)}$

（e_1 が式の場合）

我々の **TDPE**：全体像

p: プログラム, **s**: 入力の一部, **d**: 残りの入力

p と **s** の式

↓ ***eval¢***

λ**d. p(s, d)** の値

↓ ***reify***

$\mathbf{p_s}$ の式

我々の TDPE：全体像

p: プログラム, **s**: 入力の一部, **d**: 残りの入力

p と **s** の式

↓ ***preprocess***

λ**d. p(s, d)** の式′

↓ ***eval***

λ**d. p(s, d)** の値′

↓ ***reify***

$\mathbf{p_s}$ の式

- @ や + などを @′ や +′ などでおきかえる
- 静的に型つけされた言語では tag をつける

我々の **TDPE**：例

元のプログラム： **f @ (n + (s + 2))**

静的な入力： **s = 1**

動的な入力： **f** と **n**

⇓

λf. λn. f @ (n + (1 + 2)) を強正規化したい

⇓

λf. λn. f @’ (n +’ (1 +’ 2)) を ***eval*** して ***reify*** する

我々の **TDPE**：例

***reify*(λ f. λ n. f @¢ (n +¢ (1 +¢ 2)))**

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** ***reify***((λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2))) @ x)**

関数に対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** ***reify***((λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2))) @ x)**

= λ**x.** ***reify***(λ**n. x @¢ (n +¢ (1 +¢ 2)))**

（β 簡約）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** ***reify***((λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2))) @ x)**

= λ**x.** ***reify***(λ**n. x @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** λ**y.** ***reify***((λ**n. x @¢ (n +¢ (1 +¢ 2))) @ y)**

関数に対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** ***reify***((λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2))) @ x)**

= λ**x.** ***reify***(λ**n. x @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** λ**y.** ***reify***((λ**n. x @¢ (n +¢ (1 +¢ 2))) @ y)**

= λ**x.** λ**y.** ***reify***(**x @¢ (y +¢ (1 +¢ 2)))**

（β 簡約）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** ***reify***((λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2))) @ x)**

= λ**x.** ***reify***(λ**n. x @¢ (n +¢ (1 +¢ 2)))**

= λ**x.** λ**y.** ***reify***((λ**n. x @¢ (n +¢ (1 +¢ 2))) @ y)**

= λ**x.** λ**y.** ***reify***(**x @¢ (y +¢ (1 +¢ 2)))**

= λ**x.** λ**y.** ***reify***(**x @¢ (y +¢ 3))**

整数に対する +¢ の定義）

我々の TDPE：例

$$
\begin{aligned}
& \mathit{reify}(\lambda f.\ \lambda n.\ f\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \underline{\lambda x}.\ \mathit{reify}((\lambda f.\ \lambda n.\ f\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2)))\ @\ \underline{x}) \\
= {} & \underline{\lambda x}.\ \mathit{reify}(\lambda n.\ \underline{x}\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \underline{\lambda x}.\ \underline{\lambda y}.\ \mathit{reify}((\lambda n.\ \underline{x}\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2)))\ @\ \underline{y}) \\
= {} & \underline{\lambda x}.\ \underline{\lambda y}.\ \mathit{reify}(\underline{x}\ @_{\mathbb{C}}\ (\underline{y} +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \underline{\lambda x}.\ \underline{\lambda y}.\ \mathit{reify}(\underline{x}\ @_{\mathbb{C}}\ (\underline{y} +_{\mathbb{C}} 3)) \\
= {} & \underline{\lambda x}.\ \underline{\lambda y}.\ \mathit{reify}(\underline{x}\ @_{\mathbb{C}}\ (\underline{y} + 3))
\end{aligned}
$$

（式に対する $+_{\mathbb{C}}$ の定義）

我々の TDPE：例

$$
\begin{aligned}
& \mathit{reify}(\lambda f.\ \lambda n.\ f\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \lambda x.\ \mathit{reify}((\lambda f.\ \lambda n.\ f\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2)))\ @\ x) \\
= {} & \lambda x.\ \mathit{reify}(\lambda n.\ x\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \lambda x.\ \lambda y.\ \mathit{reify}((\lambda n.\ x\ @_{\mathbb{C}}\ (n +_{\mathbb{C}} (1 +_{\mathbb{C}} 2)))\ @\ y) \\
= {} & \lambda x.\ \lambda y.\ \mathit{reify}(x\ @_{\mathbb{C}}\ (y +_{\mathbb{C}} (1 +_{\mathbb{C}} 2))) \\
= {} & \lambda x.\ \lambda y.\ \mathit{reify}(x\ @_{\mathbb{C}}\ (y +_{\mathbb{C}} 3)) \\
= {} & \lambda x.\ \lambda y.\ \mathit{reify}(x\ @_{\mathbb{C}}\ (y + 3)) \\
= {} & \lambda x.\ \lambda y.\ \mathit{reify}(x\ @\ (y + 3))
\end{aligned}
$$

式に対する $@_{\mathbb{C}}$ の定義）

我々の **TDPE**：例

reify(l **f.** l **n. f @¢ (n +¢ (1 +¢ 2)))**

= ...

= l **x.** l **y.** ***reify*****(x @ (y + 3))**

= l **x.** l **y.** ***reify*****(x) @ *reify*****(y + 3)**

（**@** に対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= ...

= λx. λy. *reify*(x @ (y + 3))

= λx. λy. *reify*(x) @ *reify*(y + 3)

= λx. λy. x @ *reify*(y + 3)

（シンボルに対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= **...**

= **λx. λy. *reify*(x @ (y ± 3))**

= **λx. λy. *reify*(x) @ *reify*(y ± 3)**

= **λx. λy. x @ *reify*(y ± 3)**

= **λx. λy. x @ (*reify*(y) ± *reify*(3))**

（± に対する ***reify*** の定義）

我々の TDPE：例

***reify*(λf. λn. f @¢ (n +¢ (1 +¢ 2)))**

= ...

= λx. λy. *reify*(x @ (y + 3))

= λx. λy. *reify*(x) @ *reify*(y + 3)

= λx. λy. x @ *reify*(y + 3)

= λx. λy. x @ (*reify*(y) + *reify*(3))

= λx. λy. x @ (y + *reify*(3))

（シンボルに対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= ...

= λx. λy. *reify*(x @ (y + 3))

= λx. λy. *reify*(x) @ *reify*(y + 3)

= λx. λy. x @ *reify*(y + 3)

= λx. λy. x @ (*reify*(y) + *reify*(3))

= λx. λy. x @ (y + *reify*(3))

= λx. λy. x @ (y + 3)

整数に対する ***reify*** の定義）

我々の **TDPE**：例

reify(λ**f.** λ**n. f @¢ (n +¢ (1 +¢ 2)))**

= **...**

= λ**x.** λ**y.** ***reify***(**x @ (y** + **3))**

= λ**x.** λ**y.** ***reify***(**x**) **@** ***reify***(**y** + **3)**

= λ**x.** λ**y. x @** ***reify***(**y** + **3)**

= λ**x.** λ**y. x @ (*reify*(y)** + ***reify*(3))**

= λ**x.** λ**y. x @ (y** + ***reify*(3))**

= λ**x.** λ**y. x @ (y** + **3)**

我々の TDPE：効率の比較

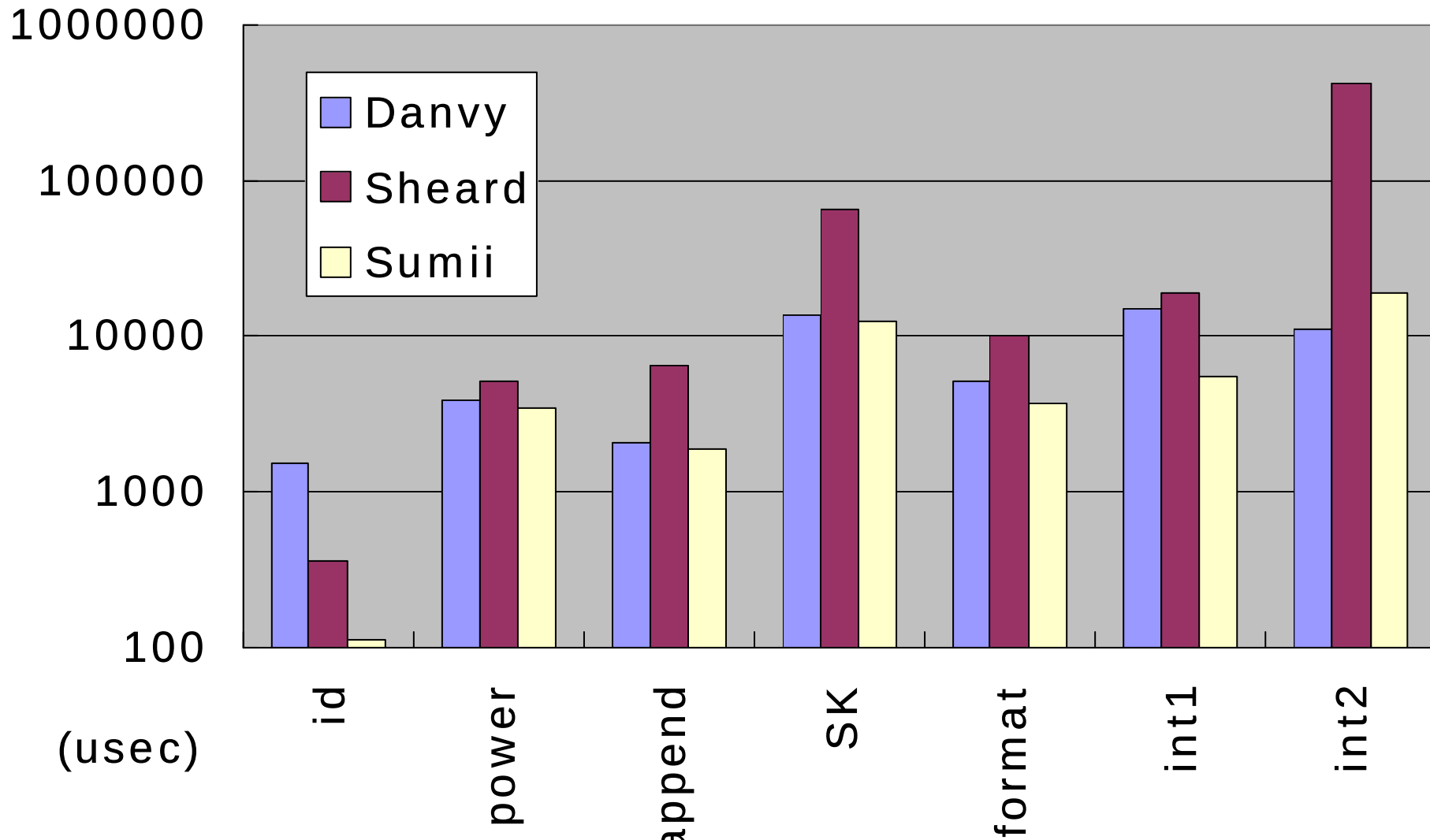
	Danvy	我々
Reflection	必要	不要
Destructor の拡張	不要	必要

どちらのほうが効率が良いのか？

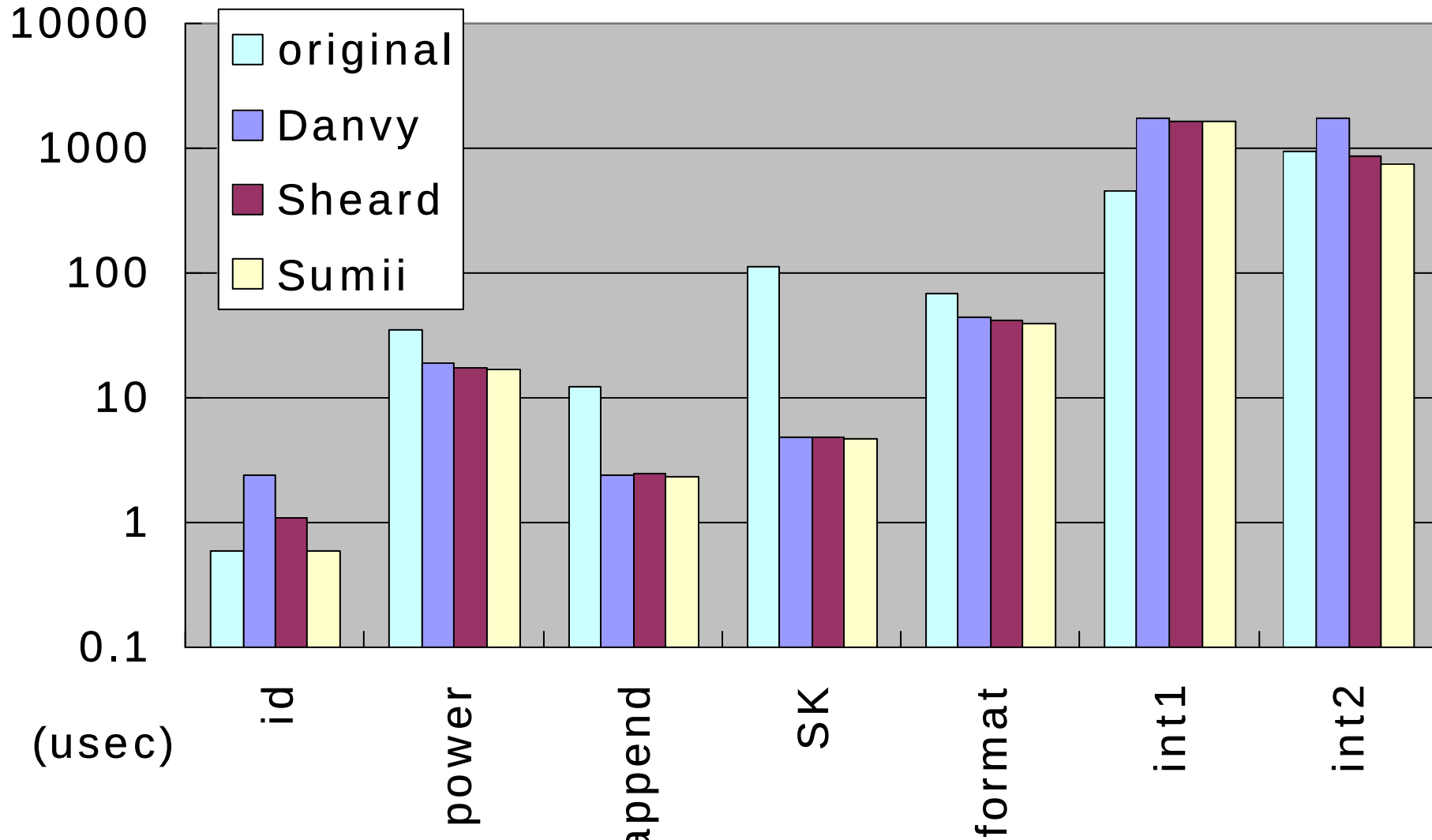
- Reflection は、Destructor の拡張と違って PE の処理だけでなく結果も悪化させる。
- 実験によれば、Destructor の拡張は Reflection よりオーバーヘッドが小さい。

⇒ 我々の TDPE のほうが効率が良い

比較実験：特化にかかる時間



比較実験：実行にかかる時間



発表の概要

- Introduction
- Danvy の TDPE
- 我々の TDPE
- 比較実験
- SDPE との関係
- 関連研究
- 結論

SDPE との関係: 全体像

単純な SDPE with HOAS & deforestation

+

HOAS からFOAS への変換

ß

我々の TDPE

SDPE との関係: HOAS

Higher-Order Abstract Syntax
[Pfenning & Elliott 88]:

object level の binding を
meta level の binding であらわす

例：

λx. λy. (x @ y) ⇒ λ(λa. λ(λb. (a @ b)))

SDPE との関係:
HOAS を用いた SDPE

$$R(\underline{\lambda}(\lambda x.\, t)) = \underline{\lambda}(\lambda y.\, R((\lambda x.\, t) \mathbin{@} y))$$

$$R(t_1 \mathbin{\underline{@}} t_2) = (\lambda x.\, t) \mathbin{@} R(t_2) \quad \text{(if } R(t_1) = \underline{\lambda}(\lambda x.\, t)\text{)}$$

$$R(t_1 \mathbin{\underline{@}} t_2) = R(t_1) \mathbin{\underline{@}} R(t_2) \quad \text{(otherwise)}$$

SDPE との関係: HOAS を用いた SDPE

$$R(\lambda(\lambda x.\, t)) = \lambda(\lambda y.\, R((\lambda x.\, t) @ y))$$

$$R(t_1 @ t_2) = (\lambda x.\, t) @ R(t_2) \quad \text{(if } R(t_1) = \lambda(\lambda x.\, t)\text{)}$$

$$R(t_1 @ t_2) = R(t_1) @ R(t_2) \quad \text{(otherwise)}$$

$$\Downarrow$$

$$\lambda\text{¢}(\lambda x.\, t) = \lambda(\lambda y.\, (\lambda x.\, t) @ y)$$

$$t_1 @\text{¢}\ t_2 = (\lambda x.\, t) @ t_2 \quad \text{(if } t_1 = \lambda(\lambda x.\, t)\text{)}$$

$$t_1 @\text{¢}\ t_2 = t_1 @ t_2 \quad \text{(otherwise)}$$

SDPE との関係: HOAS を用いた SDPE

$$R(\underline{\lambda}(\lambda x.\, t)) = \underline{\lambda}(\lambda y.\, R((\lambda x.\, t)\ @\ y))$$

$$R(t_1\ \underline{@}\ t_2) = (\lambda x.\, t)\ @\ R(t_2) \quad (\text{if } R(t_1) = \underline{\lambda}(\lambda x.\, t))$$

$$R(t_1\ \underline{@}\ t_2) = R(t_1)\ \underline{@}\ R(t_2) \quad (\text{otherwise})$$

$$\Downarrow$$

$$\underline{\lambda}^{\cent}(\lambda x.\, t) = \underline{\lambda}(\lambda y.\, (\lambda x.\, t)\ @\ y)$$

$$t_1\ \underline{@}^{\cent}\ t_2 = (\lambda x.\, t)\ @\ t_2 \quad (\text{if } t_1 = \underline{\lambda}(\lambda x.\, t))$$

$$t_1\ \underline{@}^{\cent}\ t_2 = t_1\ \underline{@}\ t_2 \quad (\text{otherwise})$$

SDPE との関係: HOAS を用いた SDPE

$$R(\underline{\lambda}(\lambda x.\, t)) = \underline{\lambda}(\lambda y.\, R((\lambda x.\, t)\ @\ y))$$

$$R(t_1\ \underline{@}\ t_2) = (\lambda x.\, t)\ @\ R(t_2) \quad (\text{if } R(t_1) = \underline{\lambda}(\lambda x.\, t))$$

$$R(t_1\ \underline{@}\ t_2) = R(t_1)\ \underline{@}\ R(t_2) \quad (\text{otherwise})$$

$$\Downarrow$$

$$\underline{\lambda}^{\cent}(\lambda x.\, t) = \underline{\lambda}(\lambda x.\, t)$$

$$t_1\ \underline{@}^{\cent}\ t_2 = (\lambda x.\, t)\ @\ t_2 \quad (\text{if } t_1 = \underline{\lambda}(\lambda x.\, t))$$

$$t_1\ \underline{@}^{\cent}\ t_2 = t_1\ \underline{@}\ t_2 \quad (\text{otherwise})$$

SDPE との関係: HOAS を用いた SDPE

$$
\begin{array}{lcll}
\boldsymbol{R}(\underline{\lambda}(\lambda \mathbf{x}.\ \mathbf{t})) & = & \underline{\lambda}(\lambda \mathbf{y}.\ \boldsymbol{R}((\lambda \mathbf{x}.\ \mathbf{t})\ @\ \mathbf{y})) & \\
\boldsymbol{R}(\mathbf{t}_1\ \underline{@}\ \mathbf{t}_2) & = & (\lambda \mathbf{x}.\ \mathbf{t})\ @\ \boldsymbol{R}(\mathbf{t}_2) & (\text{if } \boldsymbol{R}(\mathbf{t}_1) = \underline{\lambda}(\lambda \mathbf{x}.\ \mathbf{t}) \\
\boldsymbol{R}(\mathbf{t}_1\ \underline{@}\ \mathbf{t}_2) & = & \boldsymbol{R}(\mathbf{t}_1)\ \underline{@}\ \boldsymbol{R}(\mathbf{t}_2) & (\text{otherwise}) \\
 & \Downarrow & & \\
\underline{\lambda}\mathbf{¢}(\lambda \mathbf{x}.\ \mathbf{t}) & = & \underline{\lambda}(\lambda \mathbf{x}.\ \mathbf{t}) & \\
\mathbf{t}_1\ \underline{@}\mathbf{¢}\ \mathbf{t}_2 & = & (\lambda \mathbf{x}.\ \mathbf{t})\ @\ \mathbf{t}_2 & (\text{if } \mathbf{t}_1 = \underline{\lambda}(\lambda \mathbf{x}.\ \mathbf{t})) \\
\mathbf{t}_1\ \underline{@}\mathbf{¢}\ \mathbf{t}_2 & = & \mathbf{t}_1\ \underline{@}\ \mathbf{t}_2 & (\text{otherwise})
\end{array}
$$

SDPE との関係: HOAS を用いた SDPE

$$
\begin{array}{lcll}
R(\lambda x.\, t) & = & \lambda y.\, R((\lambda x.\, t)\ @\ y) & \\
R(t_1\ @\ t_2) & = & (\lambda x.\, t)\ @\ R(t_2) & (\text{if } R(t_1) = \lambda x.\, t) \\
R(t_1\ @\ t_2) & = & R(t_1)\ @\ R(t_2) & (\text{otherwise}) \\
 & \Downarrow & & \\
\underline{\lambda}\,¢(\lambda x.\, t) & = & \lambda x.\, t & \\
t_1\ \underline{@}\,¢\ t_2 & = & (\lambda x.\, t)\ @\ t_2 & (\text{if } t_1 = \lambda x.\, t) \\
t_1\ \underline{@}\,¢\ t_2 & = & t_1\ @\ t_2 & (\text{otherwise})
\end{array}
$$

⇒ @¢ の定義と一致する

SDPE との関係: HOAS からFOAS への変換

$$F(\lambda x.\ t) = \underline{\lambda y}.\ F((\lambda x.\ t)\ @\ \underline{y}) \quad \text{(where } \underline{y} \text{ is fresh)}$$

$$F(t_1\ @\ t_2) = F(t_1)\ \underline{@}\ F(t_2)$$

$$F(\underline{y}) = \underline{y}$$

⇒ ***reify*** の定義と一致する

SDPE との関係: まとめ

より複雑な SDPE を用いれば
より高度な TDPE が得られる

単純な SDPE with HOAS & deforestation

+

HOAS から FOAS への変換

ß

我々の TDPE

発表の概要

- Introduction
- Danvy の TDPE
- 我々の TDPE
- 比較実験
- SDPE との関係
- 関連研究
- 結論

関連研究

- [Danvy 96 & 97] [Sheard 97]:
  reflection のできない型には ad hoc に対応
  ⇒ 複雑で効率が悪い
- [Thiemann 96 & 99]:
  HOAS を用いた offline SDPE から
  我々と同様の変換で offline PGG を導出
- [Helsen & Thiemann 98]:
  上の offline PGG とDanvy の TDPE の
  類似性を指摘

結論

- TDPE の改良を提案
  - 広く適用できる
  - 単純で効率が良い
- 改良された TDPE と SDPE の関係を明確化

  ⇒ TDPE と SDPE の技術を融合できる

例：コードや計算の重複や消滅がなく、任意の副作用について健全な SDPE [Lawall & Thiemann 97]

◆今後の課題

- PE の停止性の保証
- 副作用を複雑に用いたプログラムの特化