Efficient Online Partial Evaluation

Eijiro Sumii
Naoki Kobayashi
University of Tokyo
({sumii,koba}@yl.is.s.u-tokyo.ac.jp)

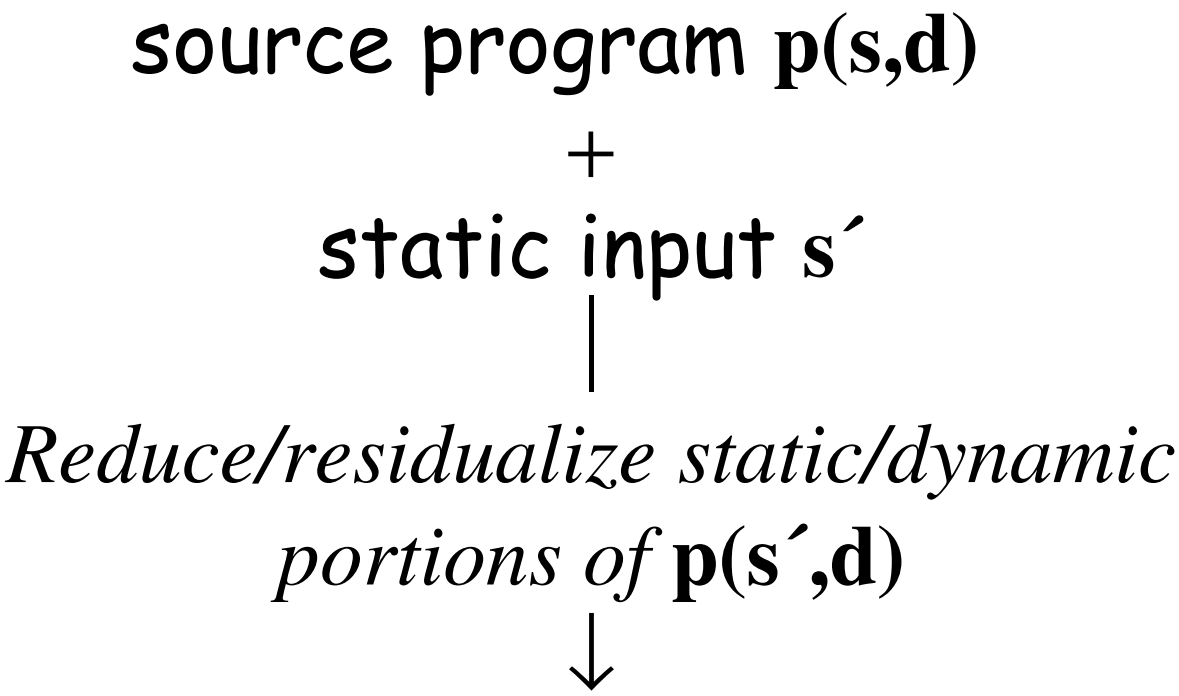
Partial Evaluation

source program **p(s,d)**

+

static input **s´**

Partial Evaluation



Partial Evaluation

source program **p(s,d)**

+

static input **s´**

|

Reduce/residualize static/dynamic portions of **p(s´,d)**

↓

specialized program $\mathbf{p_{s'}(d)}$

Partial Evaluation

source program **p(s,d)**

+

static input **s´**

|

Reduce/residualize static/dynamic portions of **p(s´,d)**

↓

specialized program $\mathbf{p_{s'}(d)}$ s.t.

$\mathbf{p(s',d) = p_{s'}(d)}$ for any dynamic input **d**

(Hopefully, the r.h.s. is faster)

Online PE and Offline PE

When to decide "static or dynamic"?

Online PE and Offline PE

When to decide "static or dynamic"?

- During specialization, with static input
  $\Rightarrow$ Online PE

Online PE and Offline PE

When to decide "static or dynamic"?

- During specialization, with static input

  $\Rightarrow$ Online PE
- Before specialization, without static input

  $\Rightarrow$ Offline PE

Merits and Demerits of Online PE and Offline PE

- Online PE is
  - more precise, but...

Merits and Demerits of Online PE and Offline PE

- Online PE is
  - more precise, but
  - less efficient
    - Takes 10-100 times as much PE time

Merits and Demerits of Online PE and Offline PE

- Online PE is
  - more precise, but
  - less efficient
    - Takes 10-100 times as much PE time
- Offline PE is
  - more efficient, but...

Merits and Demerits of Online PE and Offline PE

- Online PE is
  - more precise, but
  - less efficient
    - Takes 10-100 times as much PE time
- Offline PE is
  - more efficient, but
  - less precise
    - Requires more binding-time improvment

Our Goal and Approach

Combine

- the precision of online PE, and
- the efficiency of offline PE

by

- going back to a naive online partial evaluator, and
- optimizing it without losing its precision
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Related Work (I)

To reduce interpretive overheads

- Self-application in online PE
  [Ruf-93, Sperber-96, etc.]
  $\Rightarrow$ Cogen approach is simpler and faster

Related Work (I)

To reduce interpretive overheads

- Self-application in online PE
  [Ruf-93, Sperber-96, etc.]
  $\Rightarrow$ Cogen approach is simpler and faster
- Cogen approach to offline PE
  [Thiemann-96, etc.]
  $\Rightarrow$ We adopted it into online PE

Related Work (II)

To reduce unnecessary computations

- (Monovariant) BTA's
  - for offline PE [Henglein-91, Asai-99, etc.]
  - for online PE [Ruf-93, Sperber-96]

Related Work (II)

To reduce unnecessary computations

- (Monovariant) BTA's
  - for offline PE [Henglein-91, Asai-99, etc.]
  - for online PE [Ruf-93, Sperber-96]
- Abstract occurrence counting analysis [Bondorf-90]

$\Rightarrow$ Our analysis subsumes all of these in a single framework

Overview

- Introduction
- Related work
- *A naive online partial evaluator*
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

onpe(λx.E)ρ = ⟨*Some*(λv.onpe(E)ρ[x:=v]),

…⟩

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

onpe(λx.E)ρ = ⟨*Some*(λv.onpe(E)ρ[x:=v]),

λx$^{\Diamond}$.#$_2$(onpe(E)ρ[x:=⟨*None*, x$^{\Diamond}$⟩])⟩

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

onpe(λx.E)ρ = ⟨*Some*(λv.onpe(E)ρ[x:=v]),

λx$^{\Diamond}$.#$_2$(onpe(E)ρ[x:=⟨*None*, x$^{\Diamond}$⟩])⟩

onpe(E$_1$@E$_2$)ρ = *let* a = onpe(E$_2$)ρ *in* …

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

onpe(λx.E)ρ = ⟨*Some*(λv.onpe(E)ρ[x:=v]),

λx$^{\Diamond}$.#$_2$(onpe(E)ρ[x:=⟨*None*, x$^{\Diamond}$⟩])⟩

onpe(E$_1$@E$_2$)ρ = *let* a = onpe(E$_2$)ρ *in*

***case* onpe(E$_1$)ρ *of* …**

A Naive Online Partial Evaluator

E *(expression)* **::= $\underline{x}$ | $\underline{\lambda x}$.E | E$\underline{@}$E**

onpe : exp → env → value option × exp

(symbolic value)

onpe($\underline{x}$)ρ = ρ($\underline{x}$)

onpe($\underline{\lambda x}$.E)ρ = ⟨*Some*(λv.onpe(E)ρ[$\underline{x}$:=v]),

$\underline{\lambda x}^{\Diamond}$.#$_2$(onpe(E)ρ[$\underline{x}$:=⟨*None*, $\underline{x}^{\Diamond}$⟩])⟩

onpe(E$_1\underline{@}$E$_2$)ρ = *let* a = onpe(E$_2$)ρ *in*

***case* onpe(E$_1$)ρ *of* ⟨*Some*(v), _⟩ ⇒ v a | …**

A Naive Online Partial Evaluator

E *(expression)* **::= x | λx.E | E@E**

onpe : exp → env → value option × exp

(symbolic value)

onpe(x)ρ = ρ(x)

onpe(λx.E)ρ = ⟨*Some*(λv.onpe(E)ρ[x:=v]),

λx◊.#$_2$(onpe(E)ρ[x:=⟨*None*, x◊⟩])⟩

onpe(E$_1$@E$_2$)ρ = *let* a = onpe(E$_2$)ρ *in*

***case* onpe(E$_1$)ρ *of* ⟨*Some*(v), _⟩ ⇒ v a**

| ⟨*None*, e⟩ ⇒ ⟨*None*, e@#$_2$(a)⟩

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

Examples

- **$\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$**

$\#_2$(onpe($\lambda x.(\lambda y.y)@x$)[ ])

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathbf{onpe}(\underline{\lambda x}.(\underline{\lambda y.y})\underline{@x})[\ ])$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathbf{onpe}((\underline{\lambda y.y})\underline{@x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathrm{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\ ])$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathrm{onpe}((\underline{\lambda y}.y)\underline{@x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathrm{a} = \mathrm{onpe}(\underline{x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle]\ \mathit{in}\ \ldots)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\text{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\ ])$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\text{onpe}((\underline{\lambda y}.y)\underline{@x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \text{onpe}(\underline{x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle]\ \mathit{in}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}\ \ldots)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\text{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\text{onpe}((\underline{\lambda y}.y)\underline{@x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \text{onpe}(\underline{x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle]\ \mathit{in}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$

$\mathit{case}\ \text{onpe}(\underline{\lambda y.y})[\ldots]$

$\mathit{of}\ \ldots)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\text{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\text{onpe}((\underline{\lambda y}.y)\underline{@x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \text{onpe}(\underline{x})[\underline{x}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle]\ \mathit{in}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$

$\quad \mathit{case}\ \text{onpe}(\underline{\lambda y}.y)[\ldots]$

$\quad \mathit{of}\ \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ a = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$

$\quad \mathit{case}\ \langle \mathit{Some}(\lambda v.\text{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$

$\quad \mathit{of}\ \ldots)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathrm{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathrm{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$

$\mathit{case}\ \langle \mathit{Some}(\lambda v.\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$

$\mathit{of}\ \langle \mathit{Some}(v), _\rangle \Rightarrow v\ a$

$|\ \ldots)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathrm{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathrm{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$
$\mathit{case}\ \langle \mathit{Some}(\lambda v.\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$
$\mathit{of}\ \langle \mathit{Some}(\mathrm{v}), _\rangle \Rightarrow \mathrm{v}\ \mathrm{a}$
$\mid \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2((\lambda v.\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=v])\ \langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathbf{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathbf{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$
$\mathit{case}\ \langle \mathit{Some}(\lambda v.\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$
$\mathit{of}\ \langle \mathit{Some}(v), _\rangle \Rightarrow v\ \mathbf{a}$
$\mid \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2((\lambda v.\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=v])\ \langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathbf{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)@\underline{x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathbf{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$
$\mathit{case}\ \langle \mathit{Some}(\lambda v.\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$
$\mathit{of}\ \langle \mathit{Some}(v), _\rangle \Rightarrow v\ \mathbf{a}$
$\mid \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2((\lambda v.\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=v])\ \langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathbf{onpe}(\underline{y})[\ldots, \underline{y}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\text{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathrm{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$
$\mathit{case}\ \langle \mathit{Some}(\lambda v.\text{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$
$\mathit{of}\ \langle \mathit{Some}(v), _\rangle \Rightarrow v\ a$
$| \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2((\lambda v.\text{onpe}(\underline{y})[\ldots, \underline{y}:=v])\ \langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\text{onpe}(\underline{y})[\ldots, \underline{y}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\underline{x}^{\Diamond}\rangle$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$

$\#_2(\mathrm{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@x})[\,])$

$\rightarrow \ldots$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathit{let}\ \mathrm{a} = \langle \mathit{None}, \underline{x}^{\Diamond}\rangle\ \mathit{in}$
$\mathit{case}\ \langle \mathit{Some}(\lambda v.\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=v]), \ldots\rangle$
$\mathit{of}\ \langle \mathit{Some}(v), _\rangle \Rightarrow v\ a$
$\mid \ldots)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2((\lambda v.\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=v])\ \langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\mathrm{onpe}(\underline{y})[\ldots, \underline{y}:=\langle \mathit{None}, \underline{x}^{\Diamond}\rangle])\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\#_2(\langle \mathit{None}, \underline{x}^{\Diamond}\rangle)\rangle$

$\rightarrow \#_2\langle \ldots, \underline{\lambda x}^{\Diamond}.\underline{x}^{\Diamond}\rangle$

$\rightarrow \underline{\lambda x}^{\Diamond}.\underline{x}^{\Diamond}$

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$
  - Reduces expressions inside functions

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$
  - Reduces expressions inside functions
- ***let*** $\mathbf{f = \lambda x.1}$ ***in*** $\langle \mathbf{f\ 2, f} \rangle \rightarrow \langle \mathbf{1, \lambda x.1} \rangle$
  - Uses both the static value and the dynamic expression of $\mathbf{f}$

    cf. "both" BTA [Asai-99]

Examples

- $\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$
  - Reduces expressions inside functions
- $\mathit{let}\ \mathrm{f} = \lambda x.1\ \mathit{in}\ \langle \mathrm{f}\ 2, \mathrm{f} \rangle \rightarrow \langle 1, \lambda x.1 \rangle$
  - Uses both the static value and the dynamic expression of f
    - cf. "both" BTA [Asai-99]
- $\lambda x.1+(\mathit{if}\ \mathrm{true}\ \mathit{then}\ 2\ \mathit{else}\ x) \rightarrow \lambda x.3$
  - Doesn't require context duplication
    - cf. continuation-based PE [Lawall-Danvy-94]

Let-Insertion is Necessary

- to avoid code duplication

  $\mathit{let}\ \mathbf{f} = \lambda\mathbf{x}.\mathbf{1+2}\ \mathit{in}\ \langle \mathbf{f}, \mathbf{f} \rangle$

  $\rightarrow \mathit{let}\ \mathbf{f} = \lambda\mathbf{x}.\mathbf{3}\ \mathit{in}\ \langle \mathbf{f}, \mathbf{f} \rangle$

  rather than $\langle \lambda\mathbf{x}.\mathbf{3}, \lambda\mathbf{x}.\mathbf{3} \rangle$

Let-Insertion is Necessary

- to avoid code duplication
- to preserve semantics under side-effects

$\lambda f.1+(\mathit{let}\ x = f\ 2\ \mathit{in}\ 3)$

$\rightarrow \lambda f.\mathit{let}\ x = f\ 2\ \mathit{in}\ 4$

rather than $\lambda f.4$

Continuation-Based Let-Insertion [Lawall-Danvy-94]

Insert let-bindings by manipulating delimited continuations

- Creates a let-insertion point by delimiting a context

$$\textbf{delimit-let(e)} \equiv \textit{reset}\textbf{(e)}$$

Continuation-Based Let-Insertion [Lawall-Danvy-94]

Insert let-bindings by manipulating delimited continuations

- Creates a let-insertion point by delimiting a context

$$\textbf{delimit-let(e)} \equiv \textit{reset}(\textbf{e})$$

- Inserts a let-binding by extracting the delimited context

$$\textbf{add-let(e)} \equiv \textit{shift}(\lambda \textbf{k}.\ \underline{\textbf{let}}\ \underline{\textbf{x}}^{\Diamond} = \textbf{e}\ \underline{\textbf{in}}\ \textbf{k}\ \underline{\textbf{x}}^{\Diamond})$$

Continuation-Based Let-Insertion in Offline PE

$\underline{\lambda \mathbf{f}.(}\mathbf{1}+(\mathit{let}\ \mathbf{x} = \underline{\mathbf{f@2}}\ \mathit{in}\ \mathbf{3}))$

Continuation-Based Let-Insertion in Offline PE

λf.(1+(*let* x = f@2 *in* 3))

$\Rightarrow$ **λf.delimit-let(1+(*let* x = add-let(f@2) *in* 3))**

Continuation-Based Let-Insertion in Offline PE

$\underline{\lambda f.(1+(\mathit{let}\ x = \underline{f@2}\ \mathit{in}\ 3))}$

$\Rightarrow \underline{\lambda f.}\mathbf{delimit\text{-}let}\underline{(1+(\mathit{let}\ x =}\ \mathbf{add\text{-}let}(\underline{f@2})\ \underline{\mathit{in}\ 3))}$

$\rightarrow \underline{\lambda f.}\mathit{reset}\underline{(1+(\mathit{let}\ x =}$

$\mathit{shift}(\lambda k.\ \underline{\mathbf{let}\ x^{\Diamond} = (f@2)\ \mathbf{in}\ k\ x^{\Diamond}})\ \mathit{in}\ 3))$

Continuation-Based Let-Insertion in Offline PE

$\underline{\lambda f.(1+(\mathit{let}\ x = \underline{f@2}\ \mathit{in}\ 3))}$

$\Rightarrow \underline{\lambda f.}\text{delimit-let}\underline{(}1+(\mathit{let}\ x = \text{add-let}(\underline{f@2})\ \mathit{in}\ 3))\underline{)}$

$\rightarrow \underline{\lambda f.}\mathit{reset}\underline{(}1+(\mathit{let}\ x =$

$\mathit{shift}(\lambda k.\ \underline{\text{let}\ x^{\Diamond} = (f@2)\ \text{in}}\ k\ \underline{x^{\Diamond}})\ \mathit{in}\ 3))\underline{)}$

$\rightarrow \underline{\lambda f.}\underline{\text{let}\ x^{\Diamond} = (f@2)\ \text{in}\ (}k\ \underline{x^{\Diamond})}$

where $k[\ ] = 1+(\mathit{let}\ x = [\ ]\ \mathit{in}\ 3)$

Continuation-Based Let-Insertion in Offline PE

$\lambda f.(1+(\mathit{let}\ x = f@2\ \mathit{in}\ 3))$

$\Rightarrow \lambda f.\text{delimit-let}(1+(\mathit{let}\ x = \text{add-let}(f@2)\ \mathit{in}\ 3))$

$\rightarrow \lambda f.\mathit{reset}(1+(\mathit{let}\ x =$

$\mathit{shift}(\lambda k.\ \text{let}\ x^{\Diamond} = (f@2)\ \text{in}\ k\ x^{\Diamond})\ \mathit{in}\ 3))$

$\rightarrow \lambda f.\text{let}\ x^{\Diamond} = (f@2)\ \text{in}\ (k\ x^{\Diamond})$

where $k[\ ] = 1+(\mathit{let}\ x = [\ ]\ \mathit{in}\ 3)$

$\rightarrow \lambda f.\text{let}\ x^{\Diamond} = (f@2)\ \text{in}\ (1+(\mathit{let}\ x = x^{\Diamond}\ \mathit{in}\ 3))$

Continuation-Based Let-Insertion in Offline PE

$\underline{\lambda f.(}1+(\mathit{let}\ x = \underline{f@2}\ \mathit{in}\ 3)\underline{)}$

$\Rightarrow \underline{\lambda f.}\text{delimit-let}\underline{(}1+(\mathit{let}\ x = \text{add-let}(\underline{f@2})\ \mathit{in}\ 3)\underline{)}$

$\rightarrow \underline{\lambda f.}\mathit{reset}\underline{(}1+(\mathit{let}\ x =$

$\mathit{shift}(\lambda k.\ \underline{let}\ \underline{x^{\Diamond}} = (\underline{f@2})\ \underline{in}\ k\ \underline{x^{\Diamond}})\ \mathit{in}\ 3)\underline{)}$

$\rightarrow \underline{\lambda f.let}\ \underline{x^{\Diamond}} = (\underline{f@2})\ \underline{in}\ (k\ \underline{x^{\Diamond}})$

where $k[\ ] = 1+(\mathit{let}\ x = [\ ]\ \mathit{in}\ 3)$

$\rightarrow \underline{\lambda f.let}\ \underline{x^{\Diamond}} = (\underline{f@2})\ \underline{in}\ \underline{(}1+(\mathit{let}\ x = \underline{x^{\Diamond}}\ \mathit{in}\ 3)\underline{)}$

$\rightarrow \underline{\lambda f.let}\ \underline{x^{\Diamond}} = (\underline{f@2})\ \underline{in}\ \underline{4}$

Continuation-Based Let-Insertion in Online PE

$$\mathbf{onpe}(\underline{\mathbf{x}})\rho = \rho(\underline{\mathbf{x}})$$

$$\mathbf{onpe}(\underline{\lambda \mathbf{x}}.\mathbf{E})\rho = \langle \mathit{Some}(\lambda \mathbf{v}.\mathbf{onpe}(\mathbf{E})\rho[\underline{\mathbf{x}} := \mathbf{v}]),$$
$$(\underline{\lambda \mathbf{x}}^{\Diamond}.$$
$$(\#_2(\mathbf{onpe}(\mathbf{E})\rho[\underline{\mathbf{x}} := \langle \mathit{None}, \underline{\mathbf{x}}^{\Diamond}\rangle]))))\rangle$$

$$\mathbf{onpe}(\mathbf{E}_1 \underline{@} \mathbf{E}_2)\rho = \mathit{let}\ \mathbf{a} = \mathbf{onpe}(\mathbf{E}_2)\rho\ \mathit{in}$$
$$\mathit{case}\ \mathbf{onpe}(\mathbf{E}_1)\rho\ \mathit{of}\ \langle \mathit{Some}(\mathbf{v}), _\rangle \Rightarrow \mathbf{v}\ \mathbf{a}$$
$$\mid \langle \mathit{None}, \mathbf{e}\rangle \Rightarrow \langle \mathit{None}, \quad (\mathbf{e} \underline{@} \#_2(\mathbf{a}))\rangle$$

Continuation-Based Let-Insertion in Online PE

$$\mathbf{onpe}(\underline{\mathbf{x}})\rho = \rho(\underline{\mathbf{x}})$$

$$\mathbf{onpe}(\lambda\underline{\mathbf{x}}.\mathbf{E})\rho = \langle \mathit{Some}(\lambda \mathbf{v}.\mathbf{onpe}(\mathbf{E})\rho[\underline{\mathbf{x}}:=\mathbf{v}]),$$

$$\quad \mathbf{add\text{-}let}(\lambda\underline{\mathbf{x}}^{\Diamond}.$$

$$\quad\quad \mathbf{delimit\text{-}let}(\#_2(\mathbf{onpe}(\mathbf{E})\rho[\underline{\mathbf{x}}:=\langle \mathit{None}, \underline{\mathbf{x}}^{\Diamond}\rangle]))))\rangle$$

$$\mathbf{onpe}(\mathbf{E}_1 \underline{@} \mathbf{E}_2)\rho = \mathit{let}\ \mathbf{a} = \mathbf{onpe}(\mathbf{E}_2)\rho\ \mathit{in}$$

$$\quad \mathit{case}\ \mathbf{onpe}(\mathbf{E}_1)\rho\ \mathit{of}\ \langle \mathit{Some}(\mathbf{v}), _\rangle \Rightarrow \mathbf{v}\ \mathbf{a}$$

$$\quad\quad \mid \langle \mathit{None}, \mathbf{e}\rangle \Rightarrow \langle \mathit{None}, \mathbf{add\text{-}let}(\mathbf{e} \underline{@} \#_2(\mathbf{a}))\rangle$$

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion
- interpretive overheads
  - environment manipulation
  - syntax dispatch
  - universal domain (in typed languages)

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion
- interpretive overheads
  - environment manipulation
  - syntax dispatch
  - universal domain (in typed languages)
- unnecessary computations
  - unnecessary let-insertions
  - unnecessary *Some/None* tags
  - unused values/expressions

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion
- interpretive overheads
  - environment manipulation
  - syntax dispatch
  - universal domain (in typed languages)
- unnecessary computations
  - unnecessary let-insertions
  - unnecessary *Some*/*None* tags
  - unused values/expressions

State-Based Let-Insertion

context := λe.e

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

***let* tmp = !context *in* …**

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

let* tmp = !context *in

(context := λe.e; …

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

let* tmp = !context *in

(context := λe.e;

***let* body = e *in* …**

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

let* tmp = !context *in

(context := λe.e;

let* body = e *in

***let* head = !context *in* …**

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

let* tmp = !context *in

(context := λe.e;

let* body = e *in

let* head = !context *in

(context := tmp; …

State-Based Let-Insertion

context := λe.e

add-let(e) ≡

(context := !context ∘ λx.let $z^{\Diamond}$ = e in x; $z^{\Diamond}$)

delimit-let(e) ≡

let* tmp = !context *in

(context := λe.e;

let* body = e *in

let* head = !context *in

(context := tmp;

head body))

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion
- interpretive overheads
  - environment manipulation
  - syntax dispatch
  - universal domain (in typed languages)
- unnecessary computations
  - unnecessary let-insertions
  - unnecessary *Some*/*None* tags
  - unused values/expressions

Interpretive Approach

source program **p(s,d)**

+

static input **s´**

|

Partially-evaluate **p(s´,d)**

with an interpreter

↓

specialized program $\mathbf{p_{s'}(d)}$

Cogen Approach

source program **p(s,d)**

Cogen Approach

source program **p(s,d)**

↓

generating extension $\mathbf{cogen_p(s)}$

Cogen Approach

source program $\mathbf{p(s,d)}$

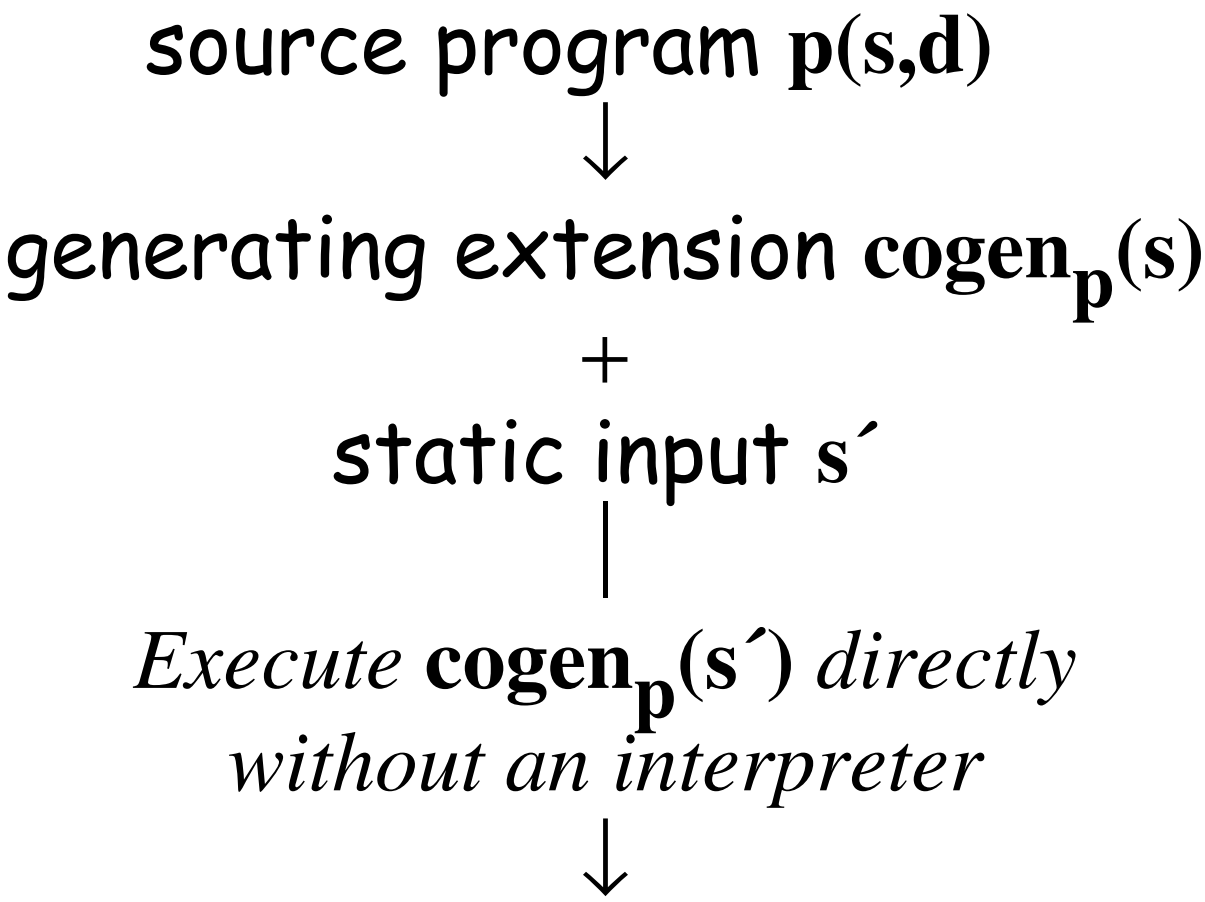
↓

generating extension $\mathbf{cogen_p(s)}$

+

static input $\mathbf{s'}$

Cogen Approach



Cogen Approach

source program **p(s,d)**

↓

generating extension $\mathbf{cogen_p(s)}$

+

static input **s´**

|

Execute $\mathbf{cogen_p(s´)}$ *directly*
without an interpreter

↓

specialized program $\mathbf{p_{s´}(d)}$

Offline-PE Combinators [Thiemann-96]

From an interpretive partial evaluator, derive a cogen

Offline-PE Combinators [Thiemann-96]

From an interpretive partial evaluator, derive a cogen using

- Higher-order abstract syntax [Pfenning-Elliott-88]
  - Substitute object-level bindings with meta-level bindings

Offline-PE Combinators [Thiemann-96]

From an interpretive partial evaluator, derive a cogen using

- Higher-order abstract syntax [Pfenning-Elliott-88]
  - Substitute object-level bindings with meta-level bindings
- Deforestation
  - Compose the syntax constructors with the partial evaluator

Online-PE Combinators

Define **abs** and **app** s.t.

onpe(**λx.E**)ρ ≈ **abs**(**λx.E′**) under ρ′

onpe($\mathbf{E_1}$**@**$\mathbf{E_2}$)ρ ≈ **app**($\mathbf{E_1'}$, $\mathbf{E_2'}$) under ρ′

Online-PE Combinators

Define **abs** and **app** s.t.

onpe($\underline{\lambda x}$.**E**)ρ ≈ **abs**(λx.**E**′) under ρ′

onpe(**E**$_1$$\underline{@}$**E**$_2$)ρ ≈ **app**(**E**$_1$′, **E**$_2$′) under ρ′

e.g.,

onpe($\underline{\lambda x}$.($\underline{\lambda y}$.y)$\underline{@}$x)[ ]

≈ **abs**(λx.**app**(**abs**(λy.y),x))

= ⟨…, $\underline{\lambda x}^{\Diamond}$.$\underline{x}^{\Diamond}$⟩

Online-PE Combinators

Define abs and app s.t.

$\mathbf{onpe}(\underline{\lambda x}.E)\rho \approx \mathbf{abs}(\lambda x.E')$ under ρ'

$\mathbf{onpe}(E_1 \underline{@} E_2)\rho \approx \mathbf{app}(E_1', E_2')$ under ρ'

e.g.,

$\mathbf{onpe}(\underline{\lambda x}.(\underline{\lambda y}.y)\underline{@}x)[\,]$

$\approx \mathbf{abs}(\lambda x.\mathbf{app}(\mathbf{abs}(\lambda y.y),x))$

$= \langle \ldots, \underline{\lambda x^{\Diamond}}.\underline{x^{\Diamond}} \rangle$

$\mathbf{abs(f)} = \langle \mathit{Some}(\lambda \mathbf{v.f\ v}), \underline{\lambda x^{\Diamond}}.\#_2(\mathbf{f}\ \langle \mathit{None}, \underline{x^{\Diamond}} \rangle]) \rangle$

$\mathbf{app(p, a)} = \mathit{case}\ \mathbf{p}\ \mathit{of}\ \langle \mathit{Some}(\mathbf{v}), _ \rangle \Rightarrow \mathbf{v\ a}$

$\quad \mid \langle \mathit{None}, \mathbf{e} \rangle \Rightarrow \langle \mathit{None}, \mathbf{e}\underline{@}\#_2(\mathbf{a}) \rangle$

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Problems of the Naive Online Partial Evaluator

Correct but inefficient, because of

- shift/reset for let-insertion
- interpretive overheads
  - environment manipulation
  - syntax dispatch
  - universal domain (in typed languages)
- unnecessary computations
  - unnecessary let-insertions
  - unnecessary *Some*/*None* tags
  - unused values/expressions

Unnecessary Computations in Online PE

- Unused values

  e.g., in $\lambda x.1+2 \rightarrow \lambda x.3$,

  $\mathbf{abs}(\lambda x.\ldots)$

  $= \langle \mathit{Some}(\lambda x.\ldots), \underline{\lambda x}^{\Diamond}.\ldots \rangle$

  can be simplified to

  $\langle (\,), \underline{\lambda x}^{\Diamond}.\ldots \rangle$

Unnecessary Computations in Online PE

- Unused values
- Unnecessary tags & unused expressions

e.g., in $(\lambda x.x)a \rightarrow a$,

$\mathbf{app(abs(\lambda x.x),a)}$

$= \mathit{case}\ \mathbf{abs(\lambda x.x)}$

$\quad \mathit{of}\ \langle \mathit{Some}(v), _ \rangle \Rightarrow v\ a \mid \langle \mathit{None}, \ldots \rangle \Rightarrow \ldots$

$= \mathit{case}\ \langle \mathit{Some}(\lambda x.x), \underline{\lambda x^{\Diamond}.\ldots} \rangle$

$\quad \mathit{of}\ \langle \mathit{Some}(v), _ \rangle \Rightarrow v\ a \mid \langle \mathit{None}, \ldots \rangle \Rightarrow \ldots$

can be simplified to

$\mathit{case}\ \langle \lambda x.x, (\ ) \rangle\ \mathit{of}\ \langle v, _ \rangle \Rightarrow v\ a$

$= \mathit{let}\ v = \lambda x.x\ \mathit{in}\ v\ a$

Unnecessary Computations in Online PE

- Unused values
- Unnecessary tags & unused expressions
- Unnecessary let-insertions

  An expression requires no let-insertion

  - if it has no side-effects, and
  - if it appears at most once in the specialized program

Type-Based Use Analysis

Count the uses of values/expressions

ρ *(raw type)* ::= $\alpha \mid \tau \rightarrow \tau \mid \tau \times \tau$

τ *(annotated type)* ::= $\rho^{(s,d)}$

Type-Based Use Analysis

Count the uses of values/expressions

ρ *(raw type)* ::= $\alpha \mid \tau \to \tau \mid \tau \times \tau$

τ *(type)* ::= $\rho^{(\mathbf{s},\mathbf{d})}$

s *(static use)* ::= **0** *(never)* | **ω** *(always)* |
T *(sometimes)*

- Whether a static value is available or not

Type-Based Use Analysis

Count the uses of values/expressions

ρ *(raw type)* ::= $\alpha \mid \tau \to \tau \mid \tau \times \tau$

τ *(type)* ::= $\rho^{(s,d)}$

s *(static use)* ::= **0** *(never)* | **ω** *(always)* |
T *(sometimes)*

- Whether a static value is available or not

d *(dynamic use)* ::= **0** *(never)* |
1 *(at most once)* | **ω** *(any number of times)*

- How many times a dynamic expression appears in the specialized program

Examples

- ***let*** **f = λx.x in f 3**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  $\Rightarrow$ **3**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  ⇒ 3
- ***let*** **f = λx.x *in* f**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  $\Rightarrow$ **3**
- ***let*** **f : *int*** $\rightarrow^{(0,1)}$ ***int*** **= λx.x *in* f**
  $\Rightarrow$ **λx.x**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  ⇒ 3
- ***let*** **f : *int*** $\rightarrow^{(0,1)}$ ***int*** **= λx.x *in* f**
  ⇒ λx.x
- ***let*** **f = λx.x *in* ⟨f 3, f⟩**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  $\Rightarrow$ **3**
- ***let*** **f : *int*** $\rightarrow^{(0,1)}$ ***int*** **= λx.x *in* f**
  $\Rightarrow$ **λx.x**
- ***let*** **f : *int*** $\rightarrow^{(\omega,1)}$ ***int*** **= λx.x *in* ⟨f 3, f⟩**
  $\Rightarrow$ **⟨3, λx.x⟩**

Examples

- ***let*** **f : *int*** $\rightarrow^{(\omega,0)}$ ***int*** **= λx.x in f 3**
  $\Rightarrow$ **3**
- ***let*** **f : *int*** $\rightarrow^{(0,1)}$ ***int*** **= λx.x *in* f**
  $\Rightarrow$ **λx.x**
- ***let*** **f : *int*** $\rightarrow^{(\omega,1)}$ ***int*** **= λx.x *in* ⟨f 3, f⟩**
  $\Rightarrow$ **⟨3, λx.x⟩**
- ***let*** **f = λx.x *in* ⟨f 3, f, f⟩**

Examples

- $\mathit{let}\ \mathrm{f} : \mathit{int} \rightarrow^{(\omega,0)} \mathit{int} = \lambda \mathrm{x}.\mathrm{x}\ \mathrm{in}\ \mathrm{f}\ 3$
  $\Rightarrow 3$
- $\mathit{let}\ \mathrm{f} : \mathit{int} \rightarrow^{(0,1)} \mathit{int} = \lambda \mathrm{x}.\mathrm{x}\ \mathit{in}\ \mathrm{f}$
  $\Rightarrow \lambda \mathrm{x}.\mathrm{x}$
- $\mathit{let}\ \mathrm{f} : \mathit{int} \rightarrow^{(\omega,1)} \mathit{int} = \lambda \mathrm{x}.\mathrm{x}\ \mathit{in}\ \langle \mathrm{f}\ 3, \mathrm{f} \rangle$
  $\Rightarrow \langle 3, \lambda \mathrm{x}.\mathrm{x} \rangle$
- $\mathit{let}\ \mathrm{f} : \mathit{int} \rightarrow^{(\omega,\omega)} \mathit{int} = \lambda \mathrm{x}.\mathrm{x}\ \mathit{in}\ \langle \mathrm{f}\ 3, \mathrm{f}, \mathrm{f} \rangle$
  $\Rightarrow \mathit{let}\ \mathrm{f} = \lambda \mathrm{x}.\mathrm{x}\ \mathit{in}\ \langle 3, \mathrm{f}, \mathrm{f} \rangle$

More Examples

- ***let*** **f = λx.x *in***
  let **g = λy.f *in***
  g

More Examples

- ***let*** **f :** $\alpha \rightarrow^{(0,1)} \alpha = \lambda x.x$ ***in***
  let **g :** $\beta \rightarrow^{(0,1)} \alpha \rightarrow^{(0,1)} \alpha = \lambda y.f$ ***in***
  g
  $\Rightarrow \lambda y.\lambda x.x$

More Examples

- *let* f : $\alpha \rightarrow^{(0,1)} \alpha$ = λx.x *in*
  let g : $\beta \rightarrow^{(0,1)} \alpha \rightarrow^{(0,1)} \alpha$ = λy.f *in*
  g
  $\Rightarrow \lambda$y.λx.x
- ***let*** **f = λx.x *in***
  let **g = λy.f *in***
  $\langle$g 1, g 2$\rangle$

More Examples

- *let* f : $\alpha \rightarrow^{(0,1)} \alpha$ = λx.x *in*
  let g : $\beta \rightarrow^{(0,1)} \alpha \rightarrow^{(0,1)} \alpha$ = λy.f *in*
  g
  $\Rightarrow$ λy.λx.x
- ***let*** **f :** $\alpha \rightarrow^{(0,\omega)} \alpha$ **= λx.x** ***in***
  let **g :** $\beta \rightarrow^{(\omega,0)} \alpha \rightarrow^{(0,1)} \alpha$ **= λy.f** ***in***
  ⟨g 1, g 2⟩
  $\Rightarrow$ ***let*** **f = λx.x** ***in*** **⟨f, f⟩**

Typing Rule

Example: λ-abstraction

$$\frac{\Gamma_0, x : \rho_1^{(s1,d1)} \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \rho_1^{(s1,d1)} \rightarrow^{(s,d)} \tau_2}$$

Typing Rule

Example: λ-abstraction

$$\Gamma \rightsquigarrow (s,d) \cdot \Gamma_0$$

$$\frac{\Gamma_0, x : \rho_1^{(s1,d1)} \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \rho_1^{(s1,d1)} \rightarrow^{(s,d)} \tau_2}$$

Typing Rule

Example: λ-abstraction

$$\Gamma \rightsquigarrow (s,d) \cdot \Gamma_0$$

$$d \neq 0 \Rightarrow s_1 \neq \omega$$

$$\Gamma_0, x : \rho_1^{(s1,d1)} \vdash e : \tau_2$$

$$\Gamma \vdash \lambda x.e : \rho_1^{(s1,d1)} \rightarrow^{(s,d)} \tau_2$$

Type Inference

- Construct the type derivation
  - assigning variables for uses, and
  - generating constraints for the variables

Type Inference

- Construct the type derivation
  - assigning variables for uses, and
  - generating constraints for the variables
- Solve the constraints
  - beginning with the most conservative approximation ($\mathbf{s} = \mathbf{T}$ and $\mathbf{d} = \omega$ for all $\mathbf{s}$ and $\mathbf{d}$), and
  - refining it with iterations (linear w.r.t. the number of use variables)

Overview

- Introduction
- Related work
- A naive online partial evaluator
- Our optimizations
  - State-based let-insertion
  - Cogen approach to online PE
  - Type-based use analysis
- Experiments
- Conclusion

Conditions

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

Applications

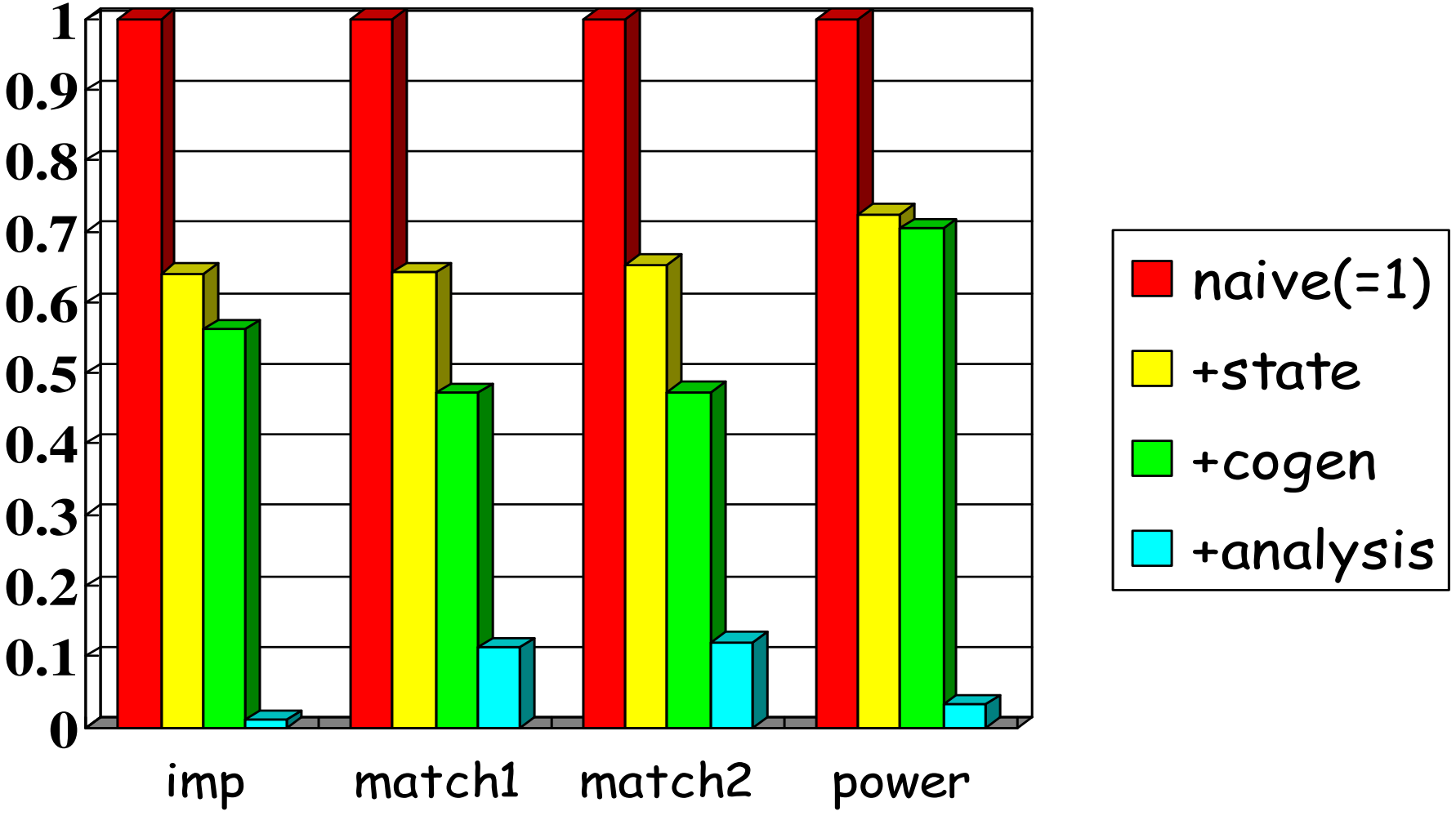
- imp

  An interpreter for a simple imperative language
- match1

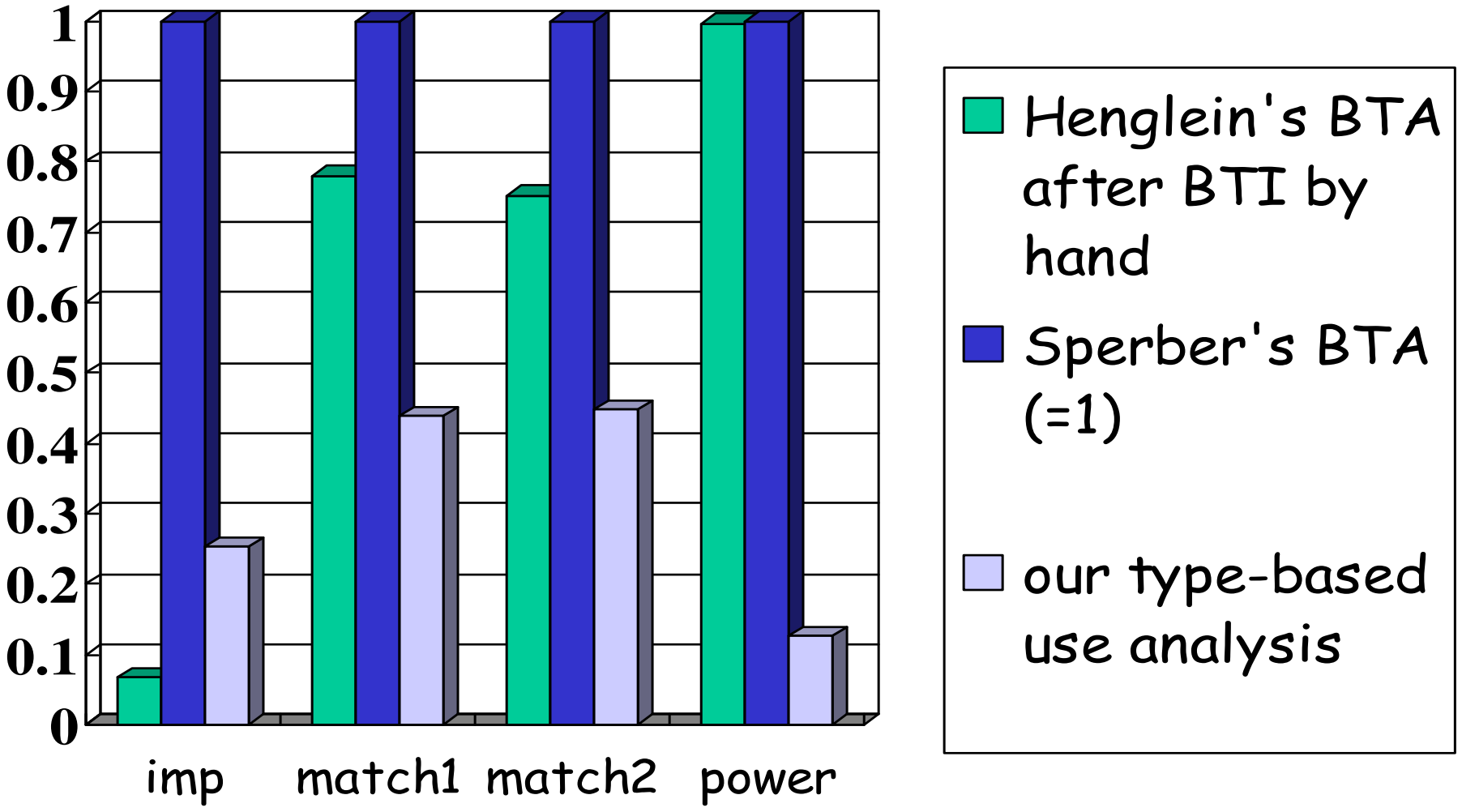
  A pattern matcher with the pattern static
- match2

  The same pattern matcher with the string static
- power

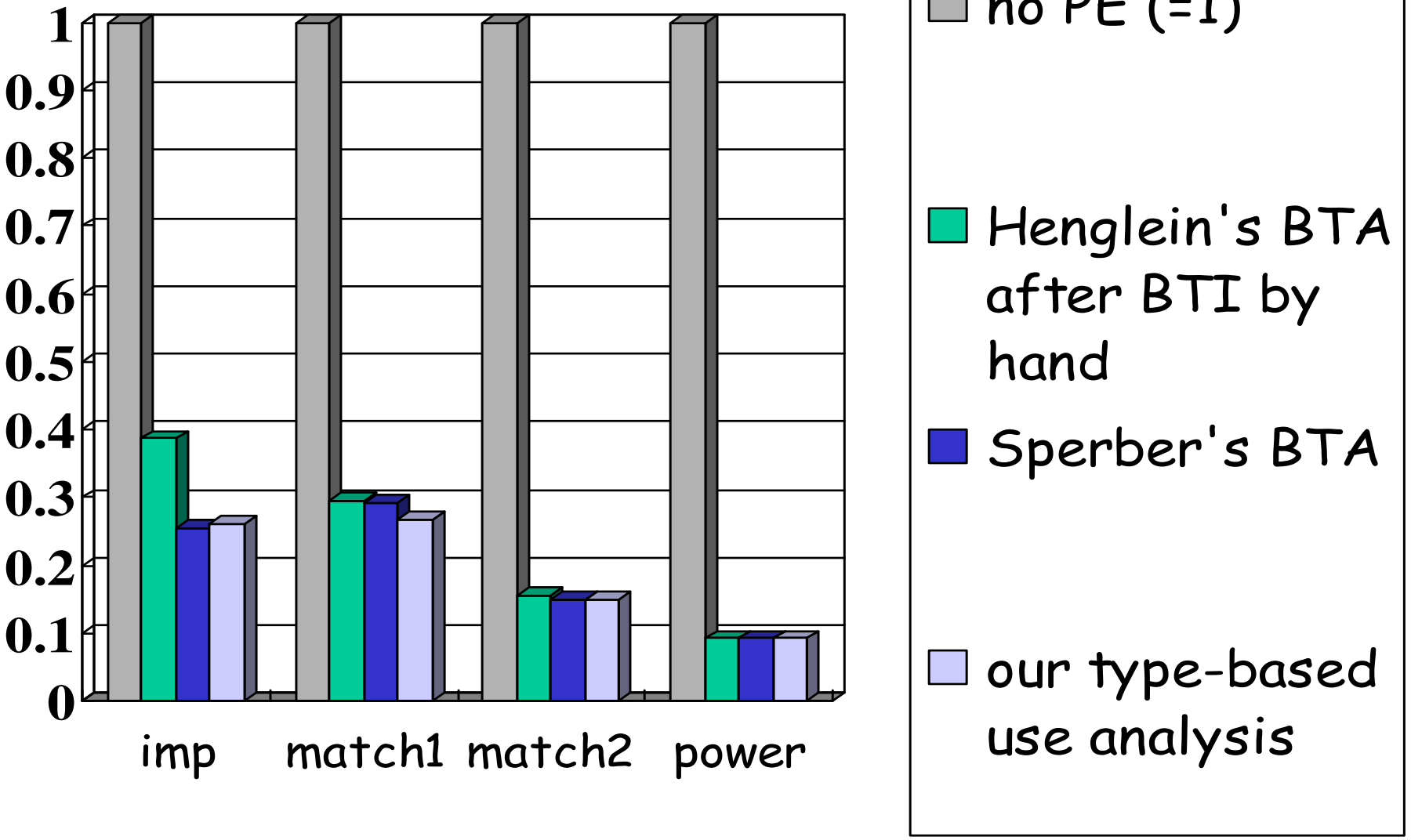
Effects of Optimizations: Time for Partial Evaluators



Comparison of BTA's: Time for Generating Extensions



Comparison of BTA's: Time for Specialized Programs



Conclusion

- We presented "hybrid" PE that combines
  - the precision of online PE, and
  - the efficiency of offline PE

Conclusion

- We presented "hybrid" PE that combines
  - the precision of online PE, and
  - the efficiency of offline PE
- Our future work includes
  - correctness proof
  - termination guarantee

Conclusion

- We presented "hybrid" PE that combines
  - the precision of online PE, and
  - the efficiency of offline PE
- Our future work includes
  - correctness proof
  - termination guarantee
- The paper to appear in PEPM'00 is available from **http://www.yl.is.s.u-tokyo.ac.jp/~sumii/pub/**